NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

REQUIREMENTS ANALYSIS AND SPECIFICATION METHODOLOGIES FOR EMBEDDED COMPUTER SYSTEMS: SURVEY AND CASE STUDY

by

Nicholas David Hunten Hammond

December, 1982

Thesis Advisor: N. R. Lyons

Approved for public release; distribution unlimited

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Requirements Analysis and Specification Methodologies for Embedded Computer Systems: Survey and Case Study		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis December, 1982 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Nicholas David Hunten Hammond		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE December, 1982 13. NUMBER OF PAGES 107
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Engineering; Software Development; Requirements Analysis; Requirements Specifications; Embedded Computer Systems; Harpoon Weapons System

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

While considerable advances have been made in the technology of software development, costs continue to rise. Research has shown that incomplete, ambiguous or inconsistent requirements specifications are a frequent cause of cost escalation and poor quality of the end product. This thesis reviews the problems in this area and their causes and examines a number of current systems and methodologies designed

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20 continued:

to better state the users' requirements. Techniques developed by the US Naval Research Laboratory for generating requirements specifications for embedded computer systems are selected for detailed examination and the results of a limited case study in the application of these techniques to a Navy weapon system are presented. These indicate that use of the techniques need not require a high degree of expertise in computer science and that they are adaptable to new systems.

Approved for public release; distribution unlimited

Requirements Analysis and Specification Methodologies for Embedded Computer Systems: Survey and Case Study

by

Nicholas David Hunten Hammond
Commander, Royal Australian Navy
B.Sc(Eng), University of London, 1967

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
December, 1982

ABSTRACT

While considerable advances have been made in the technology of software development, costs continue to rise. Research has shown that incomplete, ambiguous or inconsistent requirements specifications are a frequent cause of cost escalation and poor quality of the end product. This thesis reviews the problems in this area and their causes and examines a number of current systems and methodologies designed to better state the users' requirements. Techniques developed by the US Naval Research Laboratory for generating requirements specifications for embedded computer systems are selected for detailed examination and the results of a limited case study in the application of these techniques to a Navy weapon system are presented. These indicate that use of the techniques need not require a high degree of expertise in computer science and that they are adaptable to new systems.

TABLE OF CONTENTS

DISCLAIMER

The language in this thesis conforms to English rather than American usage. In particular, spelling is based on the Oxford English Dictionary.

I. INTRODUCTION

The high cost of computer software is a well accepted and documented fact. Due largely to reductions in hardware costs with advancing technology, the software component of overall systems cost has increased from around 20% in the 1960's to something more than 80% today. Within the software component, the cost of maintenance--changes to correct errors and to accommodate new requirements--has risen more quickly than the cost of new software development. Figure 1.1 illustrates the trends of these costs.

A. EMBEDDED SYSTEMS

This study concentrates on software for embedded computer systems. The term embedded refers to applications in which the computer is an element of a larger system. Initially, such systems were predominantly military with weapon control being the primary orientation. More recently, however, embedded computers have been used in machinery control and surveillance, data logging and other widely diverse applications.

Embedded computer system software is one of the three major software application areas. The other two are scientific computing and business software--commonly referred to as Automatic Data Processing (ADP). On a world wide basis, ADP consumes the greater proportion of the software dollar

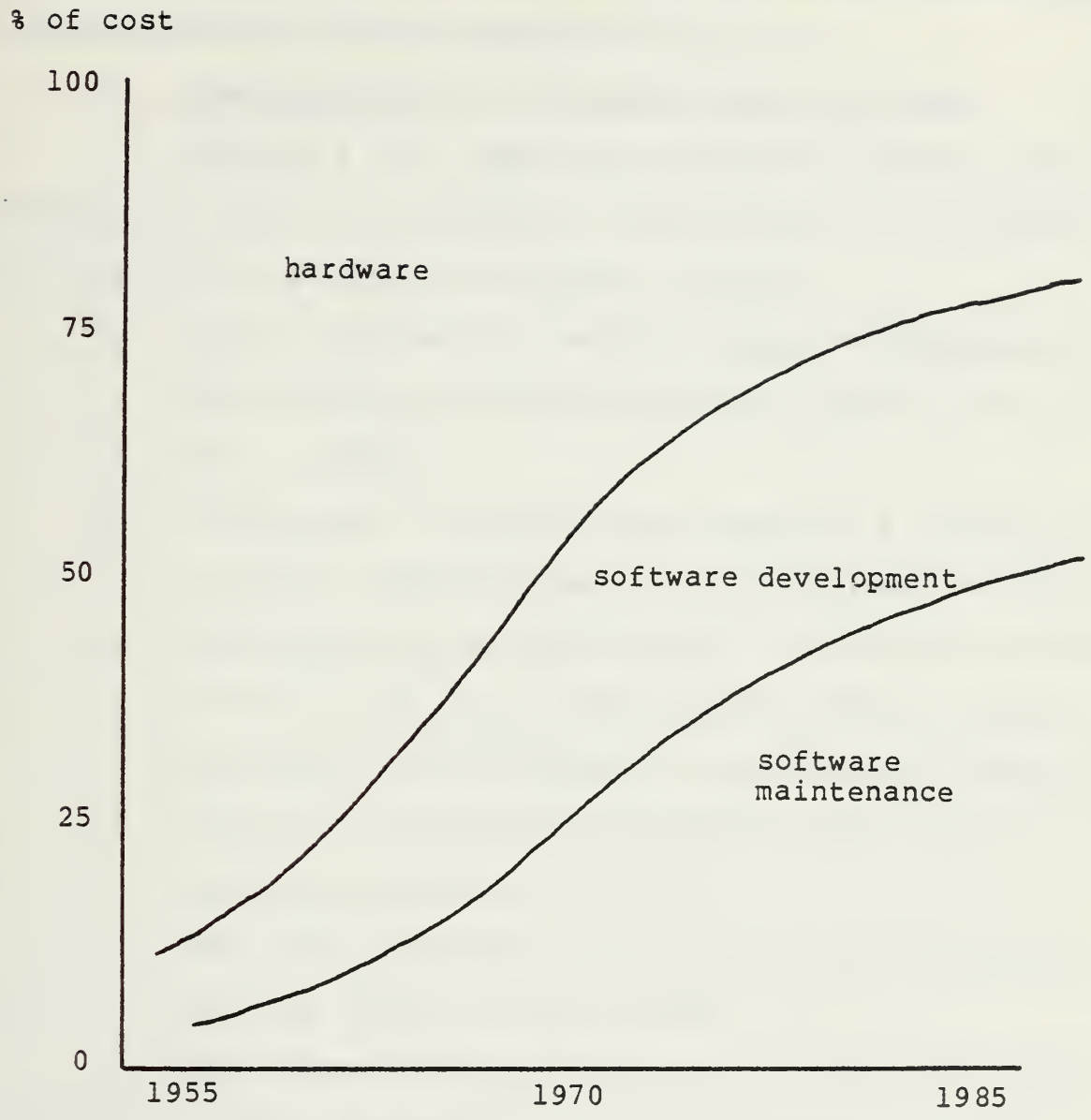

Figure 1.1
Hardware/Software Cost Trends

but within the Department of Defense (DOD), ADP only accounts for 18% of the $4.5 billion software budget with embedded system software taking 56% [1].

1. Characteristics of Embedded System Software

Software for embedded computer systems has a number of common characteristics which serve to distinguish it from ADP or scientific software [2],[22].

a. Size: programs are usually large in comparison to ADP programs, typically 100,000 + lines of code or 400 + modules.

b. Complexity: programs must control a variety of functions and interface with specialized hardware.

c. Susceptability to Modification: through the 10 to 15 year life of the associated hardware system, programs will be modified to accommodate hardware changes, new operating procedures and changes to tactical doctrine.

d. Real Time Operation: the programs must respond to external events as they occur.

The latter characteristic is probably the most critical in making embedded system software 'special'. The requirement to respond to external events means that flow of control through a program is no longer a function only of its input data and internal structure but also depends on the timing relationships of the inputs. This factor considerably increases the difficulty of design, maintenance

and testing. For example, Department of Defense costs for maintenance of embedded computer system software now exceed the amounts being expended on new development.

B. SOFTWARE ENGINEERING

During the last fifteen years, considerable advances have been made in the technology of software development. The emerging discipline of Software Engineering originated in the application of traditional engineering tools and methods to what had previously been treated as an art form. Some examples of the new techniques are the introduction of structured programming, greater use of high level languages and better program development tools such as editors, compilers and simulators.

While these techniques have resulted in significant improvements and cost reductions, a number of authorities have pointed out that over-concentration on the methods and tools of implementation rather than on the design to be implemented must result in diminishing returns for effort. Speaking of the parallel with engineering design and drafting techniques, Hoare said:

> "Perhaps our first faltering steps towards a discipline of Software Engineering are rather analogous. They are based on the discovery that a program can be designed before it is written, just as a table can be designed before it is constructed. . .
>
> "It is possible to use the most refined and accurate methods to implement the most inadequate designs. There is nothing wrong with the drawing office procedures used to construct ships which will hardly stay afloat . . ."[3]

C. REQUIREMENTS SPECIFICATION

Following this and similar calls, much work has been done in recent years on improving the early stages of the software development process--those stages which cover analysis of the requirements, preliminary and detailed design. Although new techniques have been developed and our understanding of the process improved, evidence exists that the major problems of software development still lie in these areas, particularly in the requirements analysis stage.

In a 1981 survey of 300 leading computer professionals, Thayer concluded that the problem of requirement specifications being frequently incomplete, ambiguous, inconsistent or unmeasurable was seen as the most critical of 20 major issues in software engineering project management [4]. A less formal survey [5] found that, on average, the percentage of rewriting of system specifications required during software development ranged from 15% for broad specifications to 37% for fully detailed specifications.

The cost of this problem is illustrated by a study of software errors in a large project by Boehm [6]. 64% of the errors encountered were categorised as design errors in that their correction involved changing the detailed design specification. Further, although most of the coding errors were discovered during initial test and integration 70% of the design errors were not discovered until acceptance testing

or later and thus had a disproportionate effect on both cost and schedule.

D. SCOPE

This study is directed towards a methodology for the development of requirements specifications for embedded systems software. The methodology presented is discussed in relation to the interaction between the requirements phase and other phases of the software development process.

E. ASSUMPTIONS

The reader is assumed to have a general understanding of the software development process.

F. ORGANIZATION OF THE STUDY

Chapter II presents background material on the requirements specification process, the design process and their relation to the software life cycle. This is followed in Chapter III with an overview of a number of specification techniques in current use.

Chapter IV discusses in some detail, a specific methodology developed at the US Naval Research Laboratory (NRL) and which has been applied, as part of this study, to the development of specifications for a real-world system. Chapter V provides the necessary background on the subject system and reports the results. Finally, conclusions and recommendations are given in Chapter VI.

II. BACKGROUND

A. THE SOFTWARE LIFE CYCLE

The concept of a software life cycle is a well accepted one. While there are many variants on this theme, Freeman [7] defines a five stage cycle which is representative:

a. Needs Analysis: recognition of a need by the end-user and the generation of system outlines and general requirements which will satisfy the need.

b. Specification: analysis of the need to generate specifications of system functions, objectives and constraints. May be combined with needs analysis.

c. Architectural Design: determination of overall system structure -- a 'top level' design which includes basic system relationships, hardware/software tradeoffs and major data representations.

d. Detailed Design: the fleshing out of the architectural design to include details of algorithms and data structures, lower level modularization decisions and precise interface specifications.

e. Implementation: coding, testing, integration and delivery.

f. Maintenance: all post-delivery activities including rectification of errors, modification of existing functions and addition of new functions.

This study concentrates on the first two stages but it is not possible to ignore the remainder. As Freeman points out, the stages characterize the dominant activity at each point but there is considerable interaction between phases. For example, during maintenance, the addition of a new system function will encompass all of the stages and may be regarded as a microcosm of the entire life cycle.

B. INTERACTION BETWEEN SPECIFICATION AND DESIGN PHASES

There has traditionally been much interaction between the first three stages with Needs Analysis, Specification and Architectural Design frequently forming an iterative sequence as the design is refined. In an ideal world, this would not be necessary. The specifications would be clearly and unambiguously defined and would accurately reflect the user's requirements so that the design could proceed directly. Such a situation does not appear to be generally attainable at present.

As Mc Henry and Walston [8] have pointed out, there are two primary reasons for this interaction. First, since design may be regarded as simply a restatement of one's understanding of the requirements at the next level of detail, the resulting additional scrutiny tends to expose errors and ambiguities in the original requirements specifications. Second, the knowledge gained in the system analysis leading to the requirements is essential in evaluating design alternatives.

Problems which arise from this interaction tend to be a result of the organisational structures involved in the development process. For a closely knit organization carrying on in-house development--for example, an ADP department developing software tools for internal use--there is no real penalty involved in the iterative process and there is a liklehood that a superior final design will result.

Where the users, the systems analysts and the software developers are in an arms-length relationship, as is typically the case with Defense procurements, three principal problems arise. First, the contractual relationship between analyst and developer may preclude influence of the knowledge gained in analysis on design tradeoffs. Second, the uncovering of errors and ambiguities in the requirements may require contract changes with consequent cost escalation. Third, because the developer is remote from the user, the probability of identifying problems with the user requirements, as opposed to the technical requirements, of the specification is diminished.

Mc Henry and Walston discuss the first two of these problems and conclude that the only way to avoid them is to employ procurement strategies which provide for appropriate interaction between specification and design. Those they suggest rely on contractor involvement in the analysis and specification phases as part of the contract covering

architectural design. When applicable such strategies offer advantages. In many cases however--such as when a competitive procurement is desired for design and development--requirements specifications must be developed in-house, independently of the contractor.

The third problem is less tractable still. Misunderstandings of user needs which are not resolved during the needs analysis and specification phases will tend to persist through to delivery since the user's involvement in the later stages diminishes rapidly. There appears to be no general solution to this problem, other than to develop specification methods which assist in validation of the user requirements.

C. PROBLEMS IN GENERATING SPECIFICATIONS

Distaso [9], in a 1980 survey of software management practices, concluded that obtaining satisfactory software requirements remains management's most serious challenge in bringing order to the software development process. He considered the following to be among the key elements of the problem

a. Communications barriers exist between the users, hardware designers and software designers of typical systems. The widespread and growing penetration of computer technology into other fields makes it impracticable for the average

software designer to develop familiarity with all the potential application areas. Similarly, the explosive growth of computer technology makes it difficult for experts in other fields to keep up.

b. The user's perceived needs tend to change as the system evolves. This occurs because software is an abstract concept and it is extremely difficult for users to generate precise requirements for a product which they can neither 'see' nor fully understand. As understanding progresses with system definition, the initial needs are progressively refined.

c. Scheduling difficulties, particularly with embedded systems tend to impose additional requirements changes during the specification phase. Changes in other subsystems early in the design stages have a major effect on system control software, suggesting that specification development should be delayed until the external subsystem designs have stabilized. On the other hand, the requirement for control software to support integration of these same subsystems often precludes such a delay.

D. AIMS OF A REQUIREMENTS SPECIFICATION

A software requirements specification should specify what the system must do without placing unnecessary

constraints on how this is to be achieved. Yeh and Zave [10] consider that part of the problem encountered in developing requirements is a failure by some proposed techniques to maintain adequate separation between requirements and design. Apart from the contractual difficulties mentioned above, such lack of separation may tend to make it more difficult to respond to changing user and interface requirements.

In addition to specifying what the system must do, there are a number of other uses of the requirements specification. Yeh and Zave list the following:

a. As a means of communication among users, analysts and designers.

b. To support validation of the design.

c. To control the operations and evolution of the system it specifies.

To these, Mullery [11] adds that it must allow the end-user or customer to see that the resulting system will do what they want and Heninger [12] that it should record forethought about the life cycle of the system, particularly about likely changes.

From these uses, the authors have derived a number of criteria for judging the goodness of a requirements specification.

a. Understandability: since complexity is the major issue in understanding modern software systems,

the specification must be able to decompose complexity. This may be achieved by abstraction--progressive elaboration of the system description in increasing detail; or partitioning--definition of the system in terms of sub-systems or modules.

b. Formality: to allow validation of the design and to remove ambiguity, the specification should avoid prose and present information in as formal a way as possible. The use of formal requirements allows for the possibility of automatic testing to confirm that the specification and the resulting design and implementation are logically equivalent.

c. Completeness: to avoid unnecessary system changes, the specification should be complete, explicitly stating all assumptions and constraints including non-functional constraints such as reliability and cost.

d. Modifiability: in recognition that systems undergo continuous evolution, the specification should be easy to modify. Small changes in the system environment should cause only correspondingly small changes in the system. Balzer and Goldman [13] suggest that this is best achieved by using the partitioned approach to system description and using a flexible structure which

will allow components to be easily removed or added.

E. SUMMARY

Because of communications and contractual barriers, there is frequently little interaction between end users and software system designers in most Defense software development projects. To ensure that errors and misunderstandings at the requirements analysis stage are identified prior to design, it is necessary that the software requirements specification be sufficiently clear, complete and unambiguous to allow the user to confirm that the specified system will meet his needs. In addition, the specification must meet other requirements: serving as a tool for design validation, a means of communication between analysts and implementors and a vehicle for controlling change.

III. EXISTING TECHNIQUES AND METHODOLOGIES

This chapter provides an overview of some of the more important techniques and methodologies reported in the literature which have a direct application to the first two stages of the software life cycle. These are provided to form a background against which the techniques discussed in the subsequent chapters may be judged.

Because the Software Requirements Engineering Methodology (SREM) is one of the more mature and widely used requirements developments systems and because it was specifically designed for embedded applications, it is discussed in detail and used as a vehicle to introduce concepts. Other approaches are presented more briefly and are contrasted with SREM where applicable.

A. SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY

SREM originated in the mid 1970's and was developed for use in the Ballistic Missile Defense Program [14]. The following goals were set for its development:

a. To use a structured medium or language for the statement of requirements. The properties of testability, modularity, communicability, lack of ambiguity and design freedom were to be addressed.

b. To use an integrated set of computer-aided tools to automate the process and to assure consistency,

completeness and correctness of the requirements.

c. To provide a structured approach for developing the requirements utilizing the language and validating them using the tools.

1. Processing Representation

Traditional requirements documents have adopted the functional hierarchy approach to the decomposition of complexity. All processing is divided into functions, for example: navigation, tracking, tactical display, threat response, weapons designation. These are then broken down into sub-functions in a standard hierarchy. This is the format required by MIL-STD-1679 [15] which controls software development for weapons systems for Department of Defense.

SREM uses a quite different approach based on 'messages' and 'processing paths'. A message is a data item which crosses an interface of the system under consideration. Weapon system examples range from the 'north crossing' signal -- a single bit message indicating that a particular radar beam has traversed true north, to complex streams providing large quantities of data to other systems or processors. Messages may be designated as input or output depending on the direction in which they cross the interface.

A path is the sequence of processing steps to which an input message is subjected. It may be regarded as a loop free directed graph with the input message or

stimulus at the root node, the processing steps represented by the arcs and the terminal nodes representing the response--either output messages or the alteration of stored data.

This approach is intended to overcome a number of disadvantages inherent in the function hierarchy: difficulty in being able to trace the processing required by input messages, difficulty in relating testable entities to functions which may span several subsystems, and a tendency to force the architectural design to comply with the same hierarchy and, as a result, to incorporate the same drawbacks [14].

Under SREM, the input message processing is explicit and the design of the approach was based on observations of testing of real-time software by measurement of stimulus-response performance [16]. It seems to the writer, however, that the third criticism is unjustified. While a design influenced by the SREM approach may be 'better' than one influenced by the traditional breakdown, there is no evidence that either method imposes more influence on the designer.

2. Requirements Networks

To reduce the number of paths for complex cases, paths which originate from the same input message type are integrated into a network called a Requirements Network or R-Net. The feature of the R-Net which provides path

reduction is to allow non terminal nodes to have either an OR function (only one branch path selected), an AND function (all branch paths selected) or a FOR EACH function (the single branch path repeated a number of times). The OR and FOR EACH nodes are controlled by the value of memory variables which are the result of other processing. The AND function is terminated by a re-join node which provides synchronisation -- execution does not proceed until each AND branch is complete.

The general nature of the R-Net is assured by the fact that it provides constructs analogous to those of Structured Programming. These constructs: sequential enumeration, selection and iteration have been shown by Bohm and Jacopini [17] to be sufficient to represent any process which may be represented by a standard block or flow diagram. In addition, since processing steps may be replaced by sub-nets, a hierarchical approach may be taken to allow hiding of detail not required for comprehension at a given level.

An example R-Net is shown in Figure 3.1. Some of the key features are discussed below. Processing commences on receipt of a message at the input interface (other nets may originate with an internally generated event). Since the R-Net is for a given message type, more than one instance of a particular net may exist at any one time. The

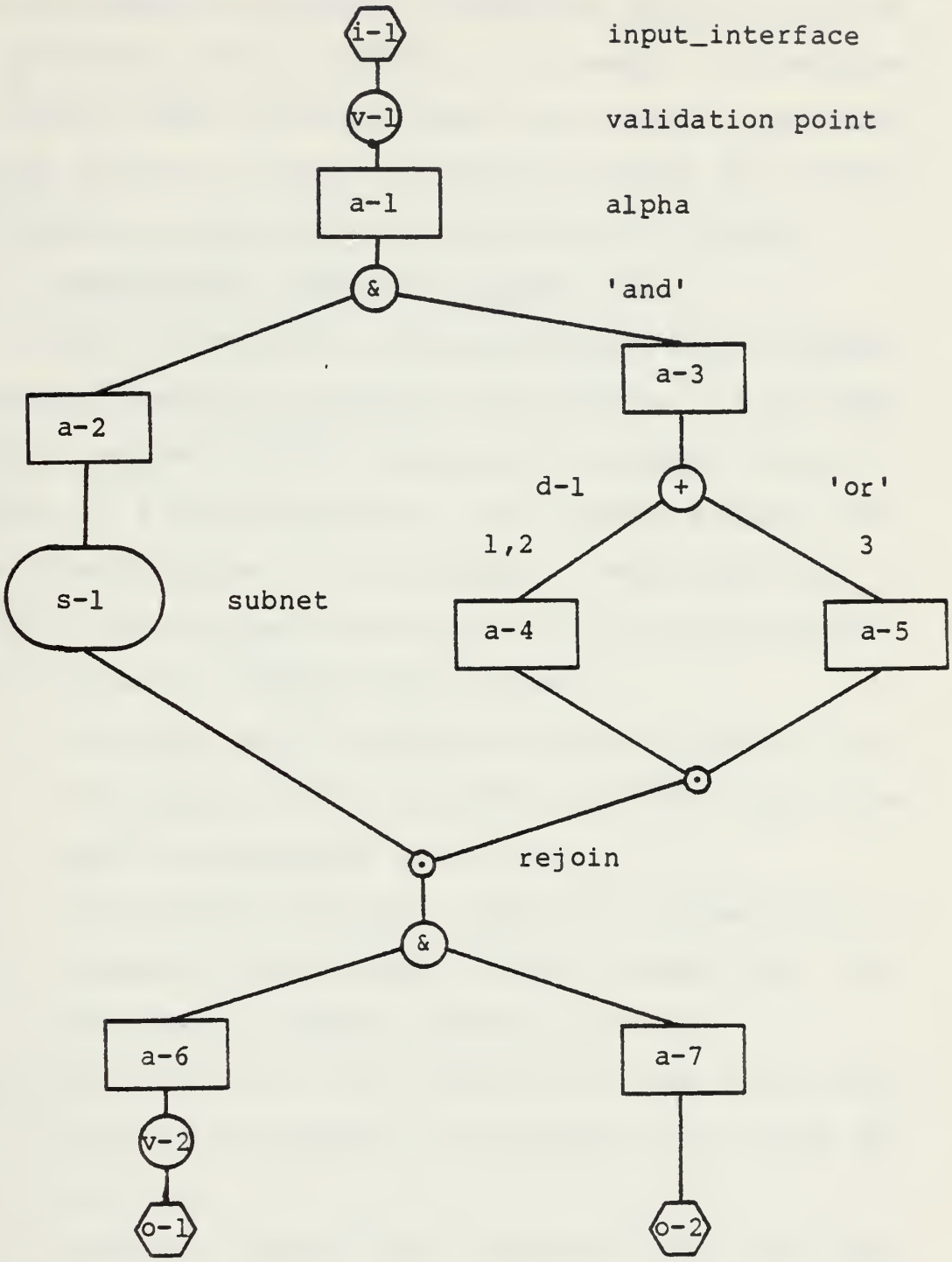

Figure 3.1
Sample R-Net (from [19])

arcs of the network represent execution paths and may be either processing steps (ALPHA's) or sub-nets. In addition to the control nodes discussed above, the network maycontain validation points at which recording of certain data during simulation may be specified for validation or testing.

3. Requirements Statement Language (RSL)

RSL is a formally designed specification language suitable for automated processing and analysis. It provides a method to express the two dimensional graphical format of the R-Nets as a one dimensional text stream suitable for machine interpretation. The language is described fully in [18] and is constructed around four primitive constructs: element, attribute, relation and structure.

a. Elements are the objects manipulated by the language and include ALPHA, R-NET and DATA (the class of conceptual data items).

b. Attributes formalize important properties of elements, for example a DATA element may have attributes of UNITS, INITIAL_VALUE etc.

c. Relations are non-comutative and state the association between two elements, eg: X_OUT .EQ. X_IN + 1.

d. Structures specify the required processing steps and may be attributed to R-Nets or sub-nets. Figure 3.2 illustrates an RSL structure embodying the the R-Net of Figure 3.1.

```
rnet: net_1
     description:
          {narrative description of function}
     refers to:
          {declarations of elements used in the r-net}
     traced from:
          {reference to originating requirement}
     structure:
          input interface: i-1
          validation point: v-1
          alpha: a-1
          do
               alpha: a-2
               subnet: s-1
          and
               alpha: a-3
               consider data: d-1
               if (1 or 2)
                    alpha: a-4
               or (3)
                    alpha: a-5
               end
          end
          do
               alpha: a-6
               validation point: v-2
               output interface: o-1
          and
               alpha: a-7
               output interface: o-2
          end
     end.
```

Figure 3.2
RSL Structure for R-NET of Figure 3.1

While the composition of the language from these primitives is fixed, the language itself is extensible in a similar manner to some programming languages such as Forth New types of elements, attributes, relationships and structures may be defined in terms of the primitives or presently defined elements.

In the original SREM, the functions allocated to the data processing sub-system and the associated performance requirements are documented as the Data Processing Subsystem Performance Requirements (DPSPR). This document is prepared by specialists in the application area and the initial SREM phase consists of its translation into RSL. Nam [19] describes a recent extension to RSL which reduces the degree of computer proficiency required of the writer and makes it feasible for at least part of the originating requirements to be entered directly into the computer by the applications specialist and translated by machine into RSL.

4. Computer Support

Computer support for RSL consists of two components. The translator and the Abstract System Semantic Model (ASSM). The primary function of the translator is to extract the primitives in the RSL document and to map them into corresponding entries in the ASSM. In addition to the normal syntax checking, the translator provides consistency checking and allows the controlled use of extensions.

The ASSM is a centralised relational database which provides an abstract model of the system described by the RSL document. The entity, relation and attributes primitives of RSL form the basic relational mapping. Entities are represented as nodes connected by relations. Attributes are connections which link entity nodes and nodes representing the attribute value or range. The fourth RSL primitive, the structure, is treated as a value which may be attributed to an R-NET or SUB-NET entity.

5. Requirements Engineering Validation System (REVS)

REVS is a collection of computer based tools which operate on the ASSM. While they are primarily aimed at the goals of completeness, correctness and consistency, they also aid in communication between customers, requirements analysts and designers. The principle areas covered are discussed briefly below:

a. Specification Analysis: Static analysis of the specification embodied in the ASSM is used to uncover internal inconsistencies and incompleteness. Since traceability to the originating DPSPR is included in the RSL specification, deficiencies in this document such as the ubiquitous TBD (To be Determined) may also be found. Typical internal inconsistencies are DATA elements which form the subject of no INPUT relation and ALPHA's which appear on no net (ie: are unused).

b. Simulation Generation: Using the model of the systems requirements specified in the ASSM, REVS is capable of automatically generating code to simulate the system. This provides a very effective means both to test the validity and consistency of the requirements and to communicate system behaviour to the users. Automatic monitoring of the validation points built in to the simulated R-Nets is provided and timing relationships are maintained to allow faithful modeling of asynchronous real-time systems. Automatic generation of the simulation code, besides saving manpower, prevents configuration management problems in maintaining traceability between simulation and requirements.

c. Interactive Graphics: REVS provides an interactive facility which allows translation between RSL structures and R-Nets. The networks may be manipulated and the modified networks retranslated. The result is that R-Nets and RSL may be used as interchangeable representations for entry into the ASSM and for communication between team members.

d. RSL Post-Processor: As a further communications aid, a post processor allows translation of the fairly cryptic RSL statements into a form more

like English. No new information is added -- just redundancy and noise -- but improved readability and user acceptance is reported [18].

6. The Methodology

RSL and R-Nets are the media used for generating requirements specifications and REVS provides the tools for their validation. Underlying these is the methodology employed by SREM -- a sequence of activities and usage necessary to generate the requirements. The steps forming this methodology are described briefly below. Alford [16] provides a full description.

a. Translation: Requirements of the DPSPR are translated into baseline requirements in RSL, R-Nets are generated and static analysis is performed to provide feedback of deficiencies in the DPSPR which is reviewed and placed under configuration management procedures at the end of the phase.

b. Decomposition: Filling out the baseline to completely specify the computational requirements of the system and to generate preliminary sub-system performance requirements. Changes in the interface specifications as sub-system design progresses may be incorporated

c. Allocation: Evaluation of path performance to allow engineering tradeoffs and generate performance requirements for the software.

d. Feasibility Demonstration: Analytical demonstration of the feasibility of critical or doubtful paths by simulation prior to design phase.

B. PROBLEM STATEMENT LANGUAGE/PROBLEM STATEMENT ANALYZER

Problem Statement Language/Problem Statement Analyser (PSL/PSA) was developed at the University of Michigan in the early 1970's [20]. While its primary application is in the problem domain of business data processing, there are a number of similarities to RSL and REVS in its general approach.

PSL describes a system model which consists of Objects and Relationships. Objects have Properties and these in turn have Property Values. Relationships connect different objects. While this is superficially similar to RSL, the difference lies in the representation of processing which, in PSL, is based on the concept of data flow rather than on the stimulus-response paths of SREM.

PSA is a collection of automated tools for performing consistency checking on the PSL model. Like REVS, it does not operate on the PSL description itself but rather on a representation of the proposed system in a database. PSA is able to perform data definition analysis, static analysis which checks the consistency of the input statements,

dynamic analysis which determines dynamic relationships between input and output and checks timing consistency and volume analysis which deals with volume of data flow. PSA provides, in addition, some management support facilities such as modification recording.

Recent enhancements to PSL/PSA have resulted in the META/GA system [21]. This allows the PSL to be varied to suit the application more closely and meets a similar objective to the extendability of RSL. The META system takes a formal meta-language description of the PSL to be used and produces a language specification. The target system is described in the new PSL and may be analysed by a Generalised Analyzer which corresponds to PSA and draws on the META Data Base to adapt to the new language. The relationships between the principal components of PSL/PSA and META/GA are illustrated in Figure 3.3.

Factors which limit the usefulness of PSL/PSA for embedded systems are the data-flow orientation of the model, which provides an inadequate representation of control paths [22], and the difficulty in expressing timing and other performance constraints [23].

C. STRUCTURED ANALYSIS AND DESIGN TECHNIQUE

The Structured Analysis and Design Technique (SADT) is an analysis and design methodology developed by Softech Inc [24]. It uses a blueprint-like graphic language (SA) to

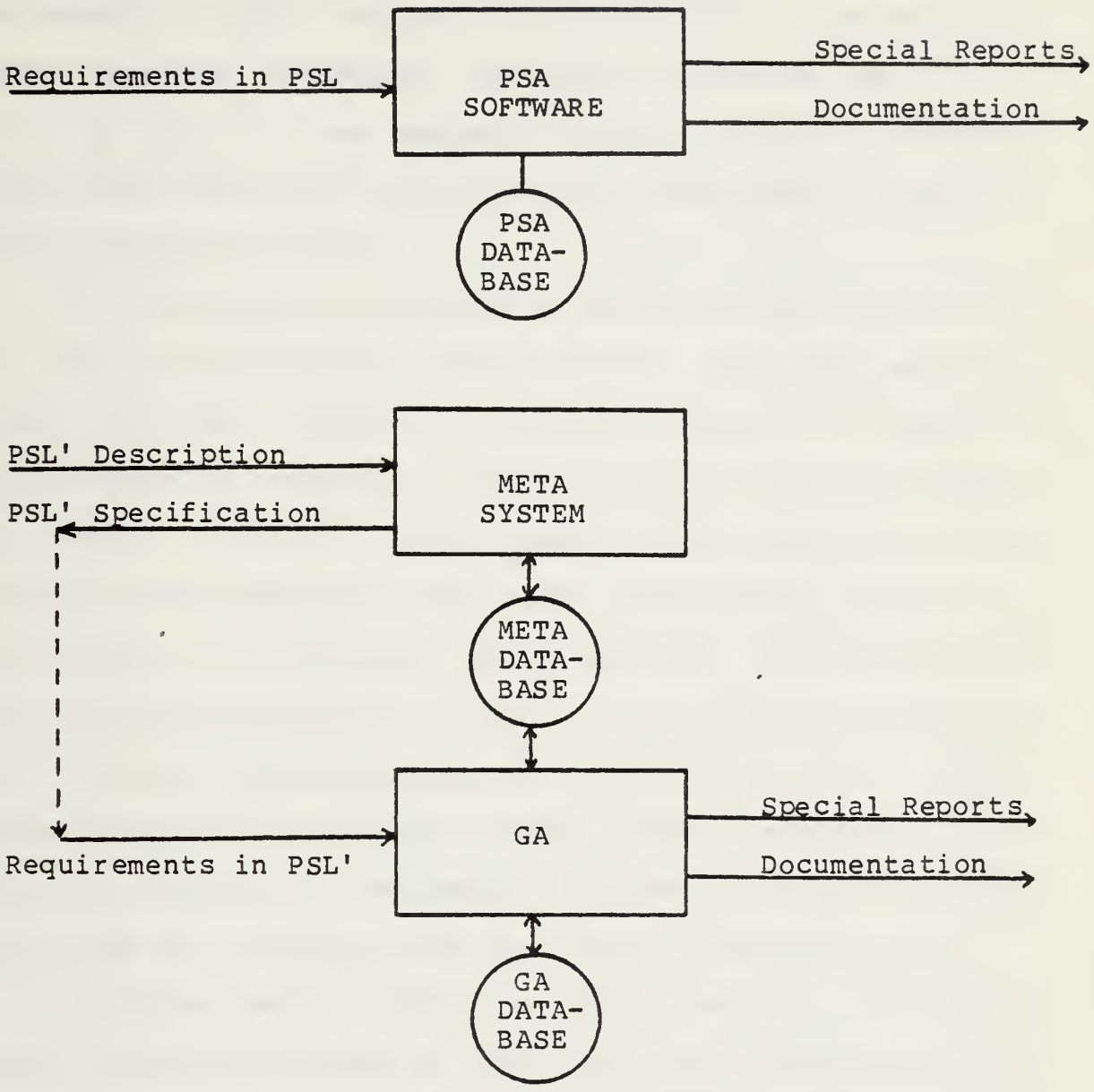

Figure 3.3
PSL/PSA and META/GA Components

represent data and control flow in a disciplined manner and its domain of application is not limited to software engineering. This language is applied using a methodology which provides management techniques including the allocation of roles to team members, regular reviews, recording development decisions and maintaining configuration control of the developing work.

The graphical primitives of SA are boxes and arrows, but the diagrams also include text to name and describe these features. A box is a 'bounded context', it may represent data or activity. Data and control flows are shown by arrows. A box must always have an arrow entering on the left side and exiting on the right, representing control or data which have undergone a transformation. An activity box must also have at least one control arrow entering the top providing the dominant constraint on the activity. Other arrows entering the top and bottom indicate additional constraints and means or mechanism. Figure 3.4 illustrates the basic SADT box structure and the method of decomposition.

A system being modeled is decomposed in a top-down manner. Boxes assigned at one level are themselves represented by lower level diagrams until the final level is reached. The result is a hierarchy of diagrams representing views of the system in progressively finer detail but allowing rapid access to the wider context of any particular aspect.

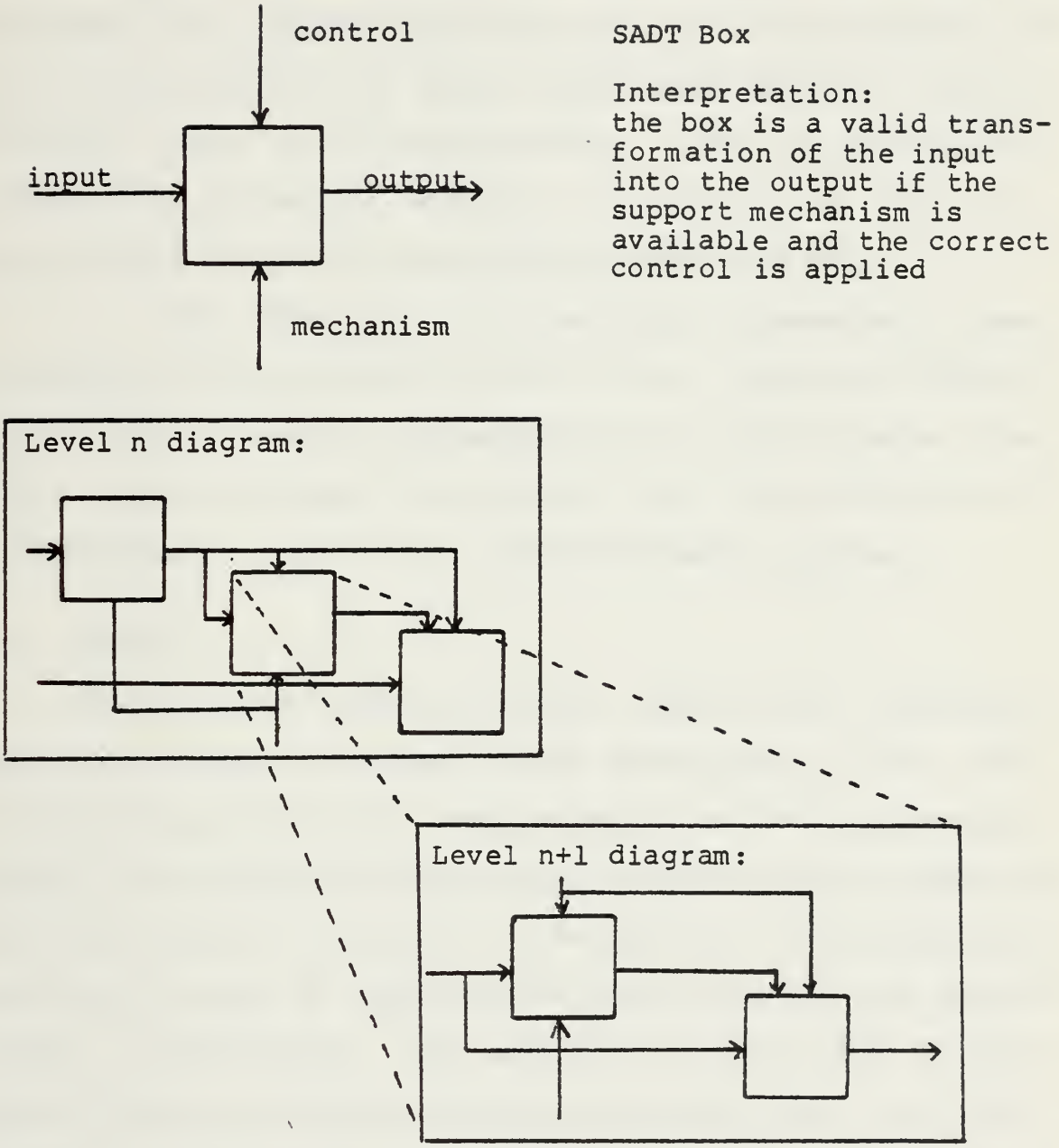

Figure 3.4 - SADT Box Diagram and Decomposition Method

SADT itself has no computer aided analysis or development tools. Ross [25] however, has pointed out that with a few exceptions, PSL and its associated database are able to represent the relationships which appear on SA diagrams and that transformation of these relationships into a machine readable form is straightforward. Such an application allows the automated analyses and support of PSA while maintaining the good communication offered by SA.

Like PSL/PSA, the processing representation used by SADT is primarily one of data flow. While the control arrows allow explicit representation of control paths, Zave [22] considers these too informal for the precision and expressiveness required for control-oriented systems.

D. PAISLEY

PAISLey is a Process Oriented, Applicative, Interpretable Specification Language under development at Bell Laboratories by Zave [22]. The language is used to develop a model of the proposed system which interacts with a model of the environment in which it is to operate. The sub-models consist of sets of asynchronous processes and the entire model is executable. The approach differs from earlier specification methods in three principal ways: explicit modeling of the environment, the use of processes (as opposed to data-flow or stimulus-response paths) as the primary specification orientation, and the fact that the models are executable.

Most system modeling approaches treat the environment as a 'black box'. Explicitly modeling it, on the other hand, offers some advantages. The complex interactions common in embedded systems can be organized around the processes which take part in them, allowing a precise, but comprehensible organization.

An explicit model also aids the anticipation of changes which will result from alterations in the environment. Since internally generated changes are limited to correction of discrepancies in the system model, these will usually form only a small proportion of the total requirements changes over the life of a typical system.

Finally, an environment model is a useful tool for requirements analysis as well as specification since it promotes appropriate questions and assists in communication with the users.

Processes are state-machine representations of autonomous computation. They are specified as a set of valid states and a 'successor function' which defines the successor state for each given state. This representation emphasises the cyclic nature of system components, in contrast to the sequential emphasis of the R-Net.

Processes operate asynchronously and communicate by means of interactions known as exchange functions which synchronize the communicating processes at the point of exchange. Figure 3.5 shows the processes in a simplified

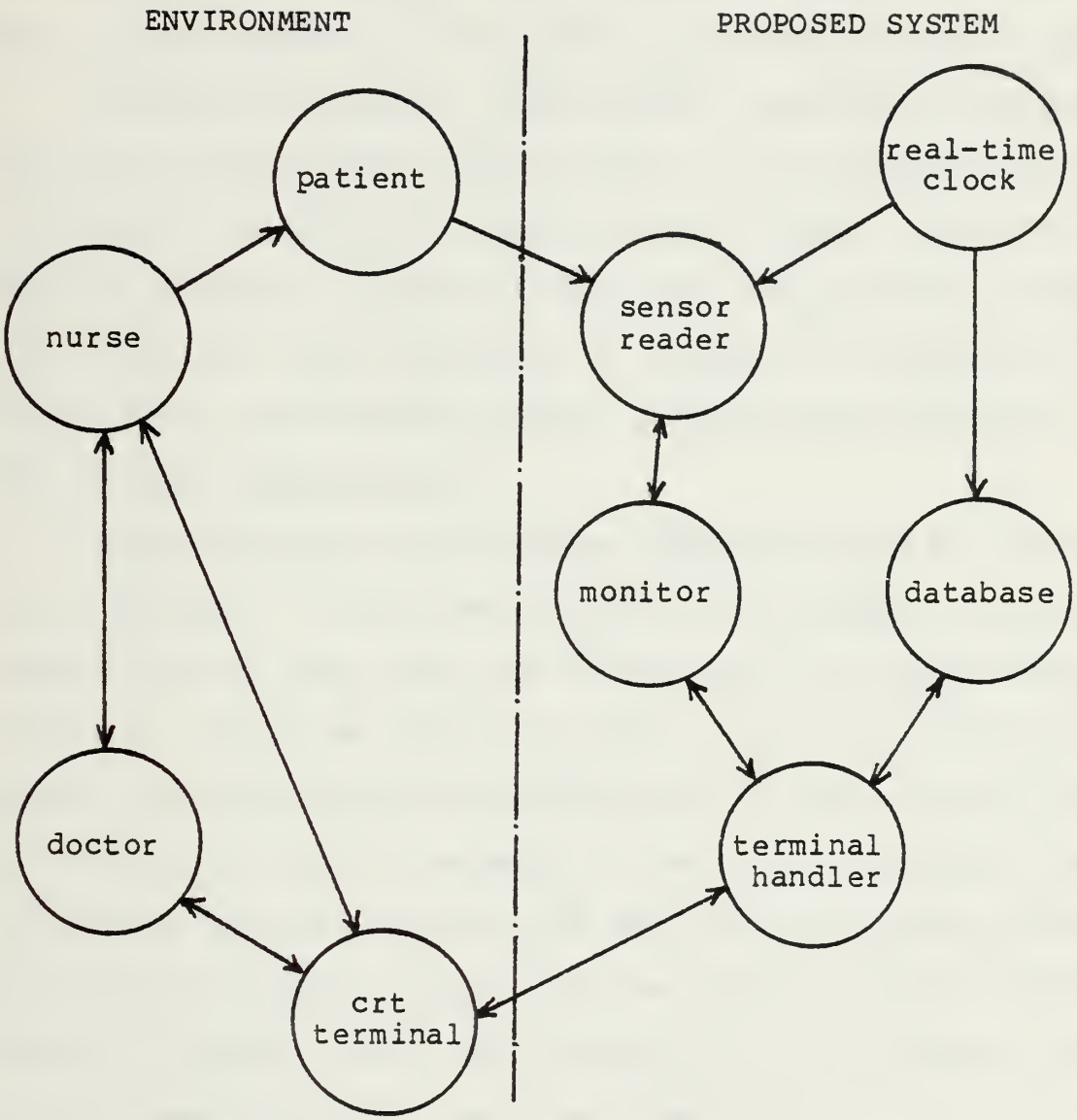

Figure 3.5
Partial Model of a Patient Monitoring System (from [22])

patient monitoring model, together with their interactions. Among the advantages of this notation are that processing of data and flow of control are integrated rather than separated as, for example, in the R-Net and DATA entities of SREM.

As does simulation under REVS, execution provides a tool for understanding and a means of validating the requirements specification and system design. Because the PAISLey language is itself executable, the need for a separate simulation code generator is avoided. In addition, the requirement for execution forces coherence and discipline on the PAISLey specification.

Disadvantages of the PAISLey approach noted by Zave are the tendency of the specification to encroach on system design and the fact that the requirement for execut-ability forces a degree of precision and a level of detail which makes the specifications difficult to understand for the untrained end user or manager. The author suggests communication using simplified aids derived from current PAISLey specifications but, unless the derivation was by machine, there is a danger that the informality so introduced could hinder communication as much as enhance it.

IV. NAVAL RESEARCH LABORATORY TECHNIQUES

The US Naval Research Laboratory and Naval Weapons Center are conducting a joint project to redesign and rebuild the Operational Flight Program (OFP) for the Navy's A-7E aircraft. While the existing program works reliably, it is expensive to maintain because its space and time margins are small, it is poorly documented and therefore not well understood by the maintenance personnel and its structure makes it difficult to change. The project will apply a number of software engineering techniques in the design of new software to alleviate these problems [26].

The objectives are to demonstrate the feasibility of using these techniques in a large scale project and to provide a useful model of their application in a Defense context to guide other systems implementors. In addition, the selection of a space and time critical program, is intended to demonstrate that the techniques do not need to impose a prohibitive processing or memory overhead.

Henninger et al [12] [27], report on the initial phase of this project--the development of a formal Software Requirements Specification which describes the external behaviour required of the OFP software without describing an OFP implementation. In addition to its primary use as the requirements document governing the remainder of the project, the specification is intended to serve as a model for

the application of the techniques used. This chapter discusses these techniques and the resulting document.

A. OBJECTIVES

A number of objectives were defined for the requirements specification. These were derived from consideration of the subsequent uses to which the document would be put and the expected readership. They recognize that the completed document will be used as a reference tool during design, implementation and maintenance, primarily by experienced programmers who already have a knowledge of the program's general purpose. It therefore must answer specific questions quickly rather than serve as a tutorial. The objectives are outlined below:

a. Specify external behaviour only. In order to avoid placing unnecessary constraints on design decisions, the requirements document should not imply any particular implementation. Rather, it should outline acceptance requirements.

b. Specify constraints on the implementation. Embedded systems software commonly must comply with externally imposed hardware interface specifications. These constraints form a legitimate part of the software reqirements specification and can be included without fear of over-influencing the design.

c. Be easy to change. As discussed in Chapter II, change is frequent in embedded systems. If the requirements document is not easy to change, its utility as a reference tool will rapidly diminish.

d. Record forethought about the life cycle of the system. As Boehm [28] and others have stated, one characteristic of quality software is its ability to be adapted to new requirements or environmental constraints without major restructuring. Clearly, some changes will be easier than others but, if the likely direction of changes is known during the design phase, appropriate provisions can be made. The appropriate time to determine this direction is during the requirements phase.

e. Characterize acceptable responses to undesired events. As for likely changes in requirements, the desired response to operator errors and hardware failures is better ascertained from the user during the requirements phase than developed ad-hoc during design and implementation.

These objectives correspond closely with those discussed in Chapter II with the exception of explicitly catering for communication between users and designers. While the ability to provide rapid answers to questions of system performance assists the analysts communications task, the format and notation conventions do not lend themselves to

casual reading and the notion of providing a tutorial introduction was specifically excluded from the requirements document's objectives.

One reason for this is that the A-7E software is what might be called 'heavily' embedded--the computer system and its software form only a small part of the entire avionics system which the user sees. Provision of user level documentation on the software alone would therefore not be useful in the absence of detailed knowledge on the part of the user of the hardware aspects of the system and how the pieces fit together.

For other systems, for example ships' combat data systems, where the software forms a greater and more central part of the whole, user level documentation on the program would be useful. In such cases, some way of re-expressing the information in a more easily comprehensible form would be desirable.

B. PROCESSING REPRESENTATION

A major difference between the A-7 specification and those designed by other methods such as SREM, is that the specification does not provide an explicit model of the system being described. The technique used derives from the State Machine of Parnas [29], one of the participants in the A-7 project. The approach is discussed briefly below -- Liskov and Zilles [30] give a more detailed treatment.

1. State Machine Model

Implicit specifications for the behaviour of an object such as a software module can be generated by identifying an abstract (possibly infinite) state machine with the object and describing the changes of state of the machine resulting from the application of the available operations.

Operations may either cause a change of state (O-operations) or return a value which is associated with some aspect of the present state (V-operations). The different states of interest in the model are therefore fully described by the set of allowable values of the V-operations and the specification is given by identifying the effect of each O-operation on all V-operations. Module properties apply to a group of operations as a whole and serve to clarify the relationship between operations.

Figure 4.1 gives the state machine specification for a simple stack data structure employing O-operations "Push" (add a data item to the top of the stack) and "Pop" (remove the top item). The state of the stack is described by the V-operations "Depth" (the number of items on the stack) and "Top" (the value of the top item). The important point to note is that the specification describes only the external behaviour required of an abstract data structure--reference to its possible implementation as a push-down stack is limited to the choice of names.

```
V-operation: TOP
     possible values: integer, initially undefined
     parameters: none
     effect: error call 1 if 'DEPTH' = 0

V-operation: DEPTH
     Possible values: integer, initially 0;
     parameters: none
     effect: none

O-operation: PUSH
     Possible values: none
     parameters: integer a
     effect: TOP = a
             DEPTH = 'DEPTH' +1

O-operation: POP
     Possible values: none
     parameters: none
     effect: error call 2 if 'DEPTH' = 0
             DEPTH = 'DEPTH' -1

Module Properties:
     The sequence PUSH(a); POP has no net effect if no
     error calls occur.
```

Figure 4.1
State Machine Specification for a Push Down Stack

Problems with state machine specifications noted by Liskov and Zilles have to do with coping with the complexity introduced when O-operations have delayed effects and allowing for changes -- a new V-operation may require changes to a large number of O-operation specifications.

2. Events and Condition Tables

In the A-7E OFP specification, the software is described as a set of functions associated with output data items. These are roughly analogous to the V-operations of state machine in that each function can be described in terms of externally visible effects and the aggregate of the functions describes the allowable states. The output values taken on by the functions are described in terms of events and conditions. A condition is a Boolean predicate which characterises some aspect of the current state of the system for a measurable period. An event defines a particular instant of time and occurs when the value of a condition changes.

Figure 4.2 shows the stack of figure 4.1 specified in terms of events and conditions. The approach retains the design freedom of the state machine model but also avoids the two disadvantages outlined above - delayed operations are easily handled by appropriate event specifications and adding additional output functions does not involve the ripple-effect observed with the O-operators.

Output data item: TOP

Condition		Events	
DEPTH > 1	PUSH(a)	X	POP
DEPTH = 1	PUSH(a)	POP	X
Action:	TOP:=a	TOP:=undefined	TOP:=*tos*

Output data item: DEPTH

Condition		Events	
DEPTH = 0	POP	X	PUSH (a)
DEPTH > 0	X	POP	PUSH (a)
Action:	Error(1)	DEPTH:=DEPTH-1	DEPTH:=DEPTH+1

Figure 4.2
Event/Condition Table Specification of Stack

3. Discussion

It is clear that the state machine and its derivatives offer greater design freedom that the techniques described in the preceeding chapter. While an implementor using a specification produced under SREM is not constrained to design modules corresponding to the Alphas and their R-Net relationships, he has a powerful incentive to do so since this arrangement has already been determined by simulation to meet the requirements. The same argument applies to PAISLey.

The A-7E approach, on the other hand, refrains from suggesting any implementation--it merely provides a set of rules against which an implementation may be tested. This need not always be a good thing, of course. As discussed in Chapter II, an iterative requirements/design sequence may be desirable in some circumstances--in these cases, a requirements specification approach which was more design oriented would be appropriate.

Zave has criticised the state machine method on the grounds that it does not permit decomposition of complexity [22]. It is certainly true that the method is unsuited to the heirarchical decomposition used in the SREM R-Net and the SADT diagrams; as the same author has pointed out, however, abstraction is the most common but is not the only available approach to decomposition [10].

The approach used by the NRL project is one of partition, in which the whole system is described as the sum of its individual parts--the functions which specify external behaviour. The widespread use of abstraction tends to suggest that it may be more generally applicable or more aligned to our natural thought processes. However, at least in the case of the A-7E specification, partitioning has provided a satisfactory solution to the problem.

C. SPECIFICATION METHODOLOGY

Heninger lists three principles used in the design of the requirements document: stating questions before trying to answer them, separating concerns to partition the task efficiently among team members and presenting the information as formally as possible to achieve precision. These have been applied consistently throughout the requirements phase of the project and form the basis of the techniques developed.

The table of contents for the A-7E document is reproduced as Figure 4.3 and illustrates the principle of separation of concerns. The heart of the specification is contained in chapters 2 to 6 and each of these deals with one aspect of the problem independently of the others. For example, chapter 2 describes the hardware interfaces without making any assumptions about the meaning of the data items concerned; chapter 3 discusses program performance in isolation of timing constraints which are covered in chapter

A-7 Software Requirements

Table of Contents

Figure 4.3
Table of Contents - A7E Requirements Document

4 and accuracy requirements in chapter 5. In addition to allowing independent work on each aspect to proceed simultaneously, this arrangement minimizes the consequential effects of changes in environment or requirements. Methods used for the various sections are outlined below.

1. Interface Specifications

Hardware interface specifications are organized by 'data item', defined as any item or input or output which changes value independently of other input or output items. Both the data items themselves and, for non numeric items, the values they may assume are given mnemonic names by which they are referenced in other sections. This allows changes in hardware-specific details to be confined to the interface specification.

As outlined above, the description of the data items contains no information on its use in the program. Switch position encoding, for example, makes reference to switch legends rather than hardware or software functions controlled. Output data items are described in terms of their effect on the associated hardware device. For each data item, all relevant software considerations such as data representation, instruction sequences for access and timing considerations are given. Figure 4.4 shows examples of input and output data item descriptions.

Input Data Item: IMS System Reliable

Acronym: /IMSREL/

Hardware: Intertial Measurement Set

Description: /IMSREL/ indicates whether or not the IMS is reliable based on a hardware self-check internal to the IMS

Characteristics of Values:

Encoding: No (0); Yes (1)

Instruction Sequence: READ 24 (Channel 0)

Data Representation: Discrete input word 5 bit 0

Output Data Item: Cursor Enable

Acronym: //CURENABL//

Hardware: Forward Looking Radar

Description: //CURENABL// directs the FLR to display Range and Azimuth cursors as specified by //CURPOS//, //CURAZCOS// and //CURAZSIN//.

Characteristics of Values:

Encoding: Off (0); On (1)

Instruction Sequence: WRITE 8 (Channel 0)

Data Representation: Discrete output word 1 bit 12

Figure 4.4
A-7 Requirements Document -
Input and Output Data Item Descriptions

2. Software Function Specifications

In the variant of the state-machine method described above, software functions are associated with output data items and are specified in terms of conditions and events. Two techniques are used to allow concise specification of complex functions, text macros and modes.

Text macros allow frequently used or complex conditions to be represented in a shorthand form. They are defined in the dictionary section and thus, only one part of the document requires alteration should the described condition change. Text macros are also useful in maintaining design freedom; where output data items depend on quantities that cannot be directly obtained from an input, a text macro can be used to specify the quantity without defining how the implementation is to derive it. For example, the text macro !ground pullup point! defines the point at which the pilot must execute a 4g pullup after weapon delivery to avoid hitting the ground. This value is obtained from altitude, speed, drag and other data but its derivation method is design/implementation decision.

Modes partition the OFP states into a number of (possibly overlapping) subsets which have applicable functions in common. They provide a shorthand means of referring to classes of conditions by the single condition 'in mode *XXX*' which is true when the system is in mode *XXX*. The selection of the modes to use may be arbitrary although

the degree to which the function descriptions are simplified will depend on the selections made. The A7-E OFP modes are based on the avionics system modes referred to in other documentation.

Modes are defined by two tables, a condition table which specifies conditions which are true when the software is in the particular mode and a transition table which lists the events which cause a transition between each combination of modes. These are partially redundant, and therefore potential sources of inconsistency, but provide the information in a convenient form. Sample mode descriptions are presented in Figure 4.5.

Software functions are classified as demand (initiated by some event) or periodic (performed repeatedly but started and stopped by an event). Functions also are specified by means of tables. Condition tables define periodic functions and specify the performance of the function under all allowable combinations of conditions and modes. Event tables show when demand functions are to be performed or when periodic functions are to be started and stopped. Figure 4.6 illustrates a periodic function description.

D. ASSESSMENT OF THE TECHNIQUES

Basili and Weiss, also working at NRL, carried out an evaluation of the A-7E Software Requirements Document by analysing changes made to the document in the 14 months after it was issued [28]. While this analysis cannot

Mode Condition Table:

(specifies the conditions that must hold in each mode -- mode names are used in the rest of the document as an abbreviation for the conjunction of the conditions in its row of the table)

Weapons Delivery Mode Conditions (partial)

Mode	/MFSW/	/FLYTOTOG/	/FLYTOTW/	/HUDREL/	Other
Nattack	$NATT$	x	x	yes	!ready station! AND NOT !reserved weapon!
Noffset	$NATTOFS$	$DEST$	NOT 0	yes	!ready station! AND NOT !reserved weapon!
a/a guns	!NO WD MFS!	x	x	x	/GUNSEL/ = Yes

Mode Transition Table:

(specifies the transitions between each mode -- rows represent the mode before transition and columns the mode after. The entries in the table are pointers to lists of conditions which control the occurrence of thge transition. A blank entry indicates that no transition is possible)

Transitions between Weapons Delivery Modes (partial)

none		N-1	N-2	N-10
Nattack	1-N	1-1		
Noffset	2-N			
a/a guns			10-2	

none	*Nattack*	*Noffset*	*a/a guns*

Figure 4.5
A-7 Requirements Document -
Mode Descriptions

Periodic Function Name: Update HUD Magnetic Heading

Modes in which function required:

Alignment:	all
Navigation:	all
Test:	*Grtest*

Initiation/Termination Events: None - always performed

Output Data Items: //MAGHDGH//

Output Description:

Condition Table 4.3.6-a Magnetic Heading

MODES	CONDITIONS	
All Alignment and Navigation Modes except *IMS fail*	Always	X
IMS fail	(NOT /IMSMODE/=$Offnone$)	/IMSMODE/=$Offnone$
//MAGHDGH//	angle defined by /MAGHCOS/ and /MAGSIN/	0 (North)

Figure 4.6
A-7 Requirements Document -
Periodic Function Descriptions

provide a direct comparison with documents produced by other methods, it does provide sufficient data to allow a subjective assessment of the degree to which some of the objectives of the project have been met.

The use made of the document was inferred from an analysis of the source of changes. This indicated that it was in heavy use as a design reference as intended. There is no current application as a maintenance reference for the new OFP at this stage of the project but there is evidence that the document is being used by maintainers of the existing software.

In the analysis period, 79 errors in the requirements document were discovered. The predominant causes were incorrect fact (37%), omissions (24%) and clerical errors (23%). Errors involving internal inconsistency and ambiguity were considerably less common at 10% and 4% respectively. The frequency of factual errors is surprising in the light of the fact that the requirements document was validated against the existing OFP by experienced personel before it was published.

The majority of the errors (74%) were discovered in sections 2 and 4, which deal with the hardware interfaces and the software functions respectively. In section 2 most of the errors were omissions and in section 4 the majority were factual.

These sections are the two most directly concerned with the external interfacing and behaviour of the program, and their development requires significant interaction with users and hardware specialists. It is tempting to attribute the error incidence in these sections to the lack of communication aids discussed above. However, as the authors of the survey point out, it may also be influenced by the use being made of the document at this stage of the project and a full analysis of errors by section should await completion of implementation.

Effort to correct the errors totalled eleven man-weeks with 94% requiring less than one man-day. The high total figure was caused by a factual error in the specification of a coordinate system which required considerable research to determine the correct information and contributed more than 50% of the total. Considering the total effort involved in producing the document (75 man-weeks) and in software development during the analysis period (122 man-weeks), the authors consider that the effort to maintain the document was acceptably low. Ease of change appears to be confirmed by the fact that 85% of the changes made to the document affected only one section. The major error referred to above was in this category.

No case was discovered where implementation facts were incorrectly given by the requirements document, implying that the objective of design freedom has been met. The

question of whether the remaining objectives, to do with the forecasting of change and the specification of error responses, have been met will remain open for some time. The data recording on which Basili and Weiss based their results is continuing and answers to these questions should become available.

V. CASE STUDY - THE HARPOON WEAPONS SYSTEM

As outlined in the last chapter, the techniques developed during the NRL study offer an attractive alternative, in certain circumstances, to other methods in use--both ad hoc and formalized. The A-7E project, however, was staffed by highly qualified members and was duplicating an existing program--all the answers on required external program behaviour were available by testing the response of the original software.

In an attempt to gain experience with the techniques in another application, a limited case study was conducted by using them in the development of a partial software requirements document for a replacement weapons control console for the Harpoon weapons system. This chapter provides some necessary background details on that system and discusses the application of the NRL techniques in this context.

A. EXISTING HARPOON WEAPON SYSTEM

Harpoon is an antiship missile which may be launched from a variety of platforms--aircraft, submarines, and surface ships. It has an all-weather capability and provides a means of mounting an attack against a surface target from beyond horizon range. The missile incorporates a booster section for ship and submarine launch but, beyond this, variations for the different platform types are minor.

1. Overview

Prior to launch, targetting parameters are passed to the missile by the launch platform. These control the flight path, search patterns and modes and desired terminal manouevres. Once it has been launched, the missile is autonomous. It flies down the selected flight path until a target is detected in the search area by radar or passive detection of electromagnetic radiation. It then descends to near sea level en route to the target, to reduce the probability of its being detected or successfully counter-attacked by the target. Finally it performs the selected terminal manoeuvering and attack.

For ship Harpoon weapon system installations, the preparation and launch of missiles is accomplished by the Harpoon Shipboard Command and Launch Control Set (HSCLCS). This subsystem receives as inputs targeting data, ship's motion data, and Harpoon missile and launcher cell status reports. HSCLCS provides the following control functions:

a. Missile initialization and power up.
b. Passing of targetting parameters to missile.
c. Launcher alignment (in systems with trainable launchers).
d. Missile launch.
e. Monitoring of missile and launcher cell status.

Figure 5.1 illustrates the HSCLCS in block diagram form. The components of immediate interest in the system

are the Harpoon Control Console (HCC) and the associated Weapon Control Indicator Panel (WCIP). The HCC provides the control function for the entire HSCLCS as well as the interface to the ships Weapon Control System (WCS)--an information system used to support tactical decision making. The WCIP provides the interface for the Harpoon weapon system operator. The remainder of the equipment is used to interface the HCC to the launcher, missile and the ships environmental sensors.

The HCC control functions are provided by the Data Processor Computer (DPC) -- an early 16 bit microcomputer with limited memory. The DPC utilizes an assembly language program to provide the following functions:

a. Receipt of target range and bearing and recommended engagement parameters from WCS (or from WCIP in local mode).

b. Receipt of commanded engagement parameters from WCIP.

c. Launch envelope parameter validation.

d. Missile command generation.

e. Prelaunch testing, sequencing and timing.

f. System 'housekeeping'.

The WCIP provides for operator entry of engagement parameters and displays status information to the Harpoon

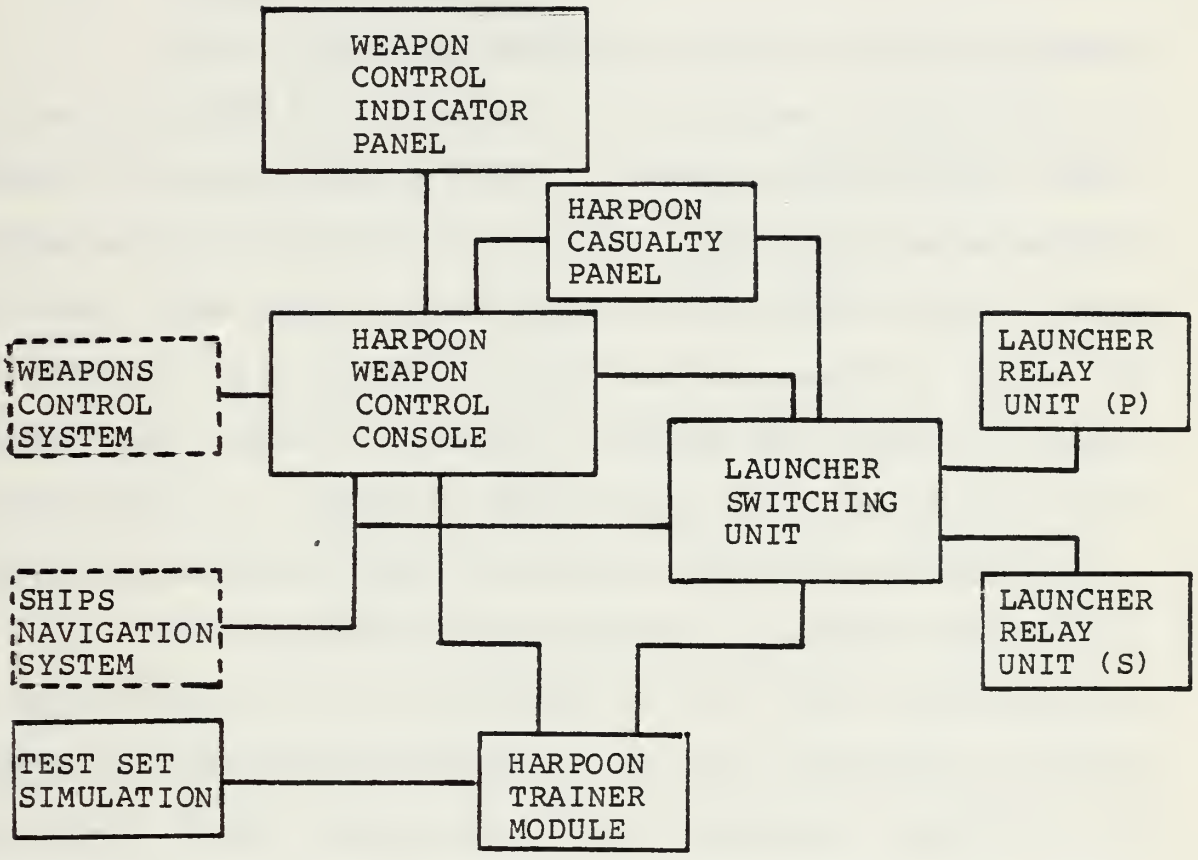

Figure 5.1
HSCLCS - Block Diagram

system operator during formulation of the fire control solution. If target parameters are unavailable from WCS, they may be manually entered by the operator. Details of the WCIP operator controls are given in Figure 5.2.

2. Engagement Selection

In the original HSCLCS, control of the engagment to be conducted by the missile is limited to selection of Range & Bearing Launch (RBL) or Bearing Only Launch (BOL) modes and to selection of a small, medium or large search pattern. BOL mode is used when no reliable target range information is available -- the missile searches along the flight path between specified minimum and maximum ranges. In RBL mode, the pattern determines the size of the 'box' about the selected target position which will be searched.

To accommodate improvements in missile capability, a modification was introduced in late 1981, providing a number of additional functions. The more important of these are small target selection, search expansion selection and waypoint entry. All of these provide means of increasing the selectivity of the missile in an environment where there are a number of potential targets.

Target selection conditions the target detection logic of the missile to respond to different sized radar returns and modifies the relative destruction probabilities of different size targets within the search box. Search expansion selection performs a similar function by setting

PHM WEAPON CONTROL INDICATOR PANEL

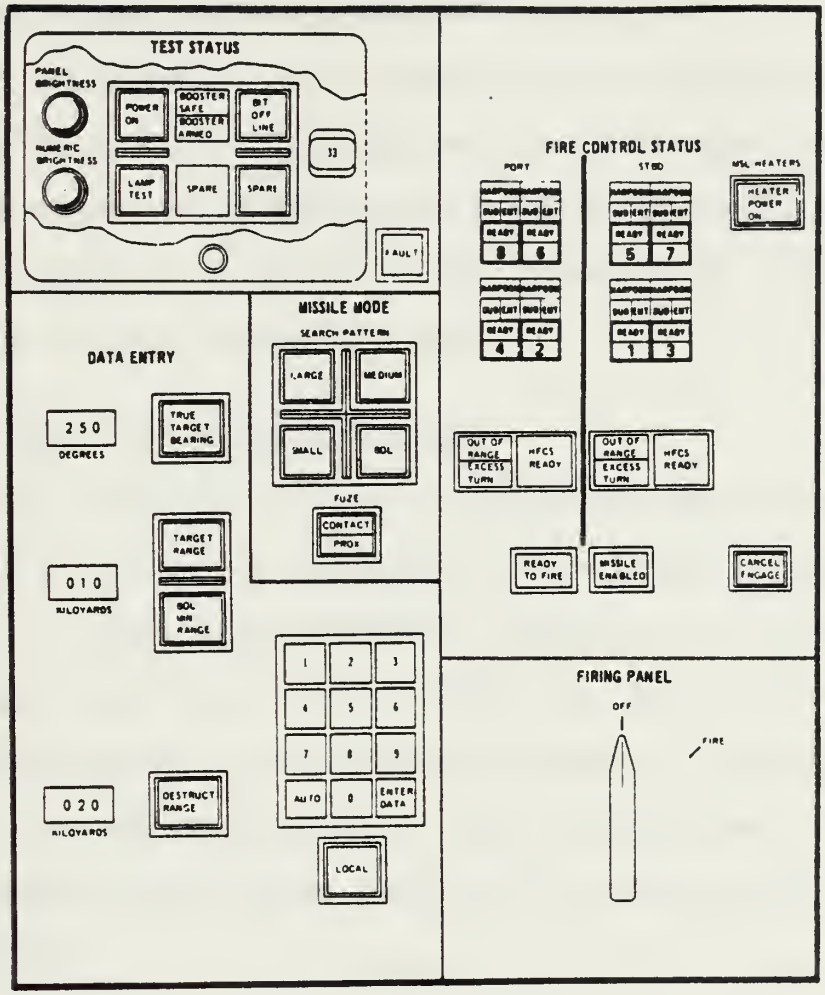

Figure 5.2
Existing WCIP

the initial direction in which the seeker head moves in the search pattern. Waypoints are positions through which the flight path passes en route to the target and allow the path to be selected to avoid neutral or friendly ships and to approach the target from a preferred bearing.

Other changes are aimed at improving the kill probability of the missile against well defended targets. They provide means of initializing a salvo of missiles with different engagement parameters and timing their firing for simultaneous arrival at the target.

B. DEFICIENCIES OF EXISTING SYSTEM

The major drawback of the existing system is that the information provided is inadequate for proper engagement planning. In ships where the WCS function is provided by the Naval Tactical Data System (NTDS), a limited amount of automated assistance is available but much of the planning must rely on manual calculation. In an enviromnent of tight time constraints and extreme stress, the potential for human error is large.

While NTDS could potentially provide the required information in a suitable form, there are a number of difficulties in this approach. First; not all Harpoon fitted ships are also equipped with NTDS. Second; in most ships, NTDS facilities are fully committed and the Harpoon engagement control function could only be accommodated at the expense of some other capability.

The second deficiency is that the full capabilities of the flight vehicle are unable to be used due to an inability of the HSCLCS to set the required pre-launch parameters. The original HSCLCS assumes a fixed bearing flight to the target position with limited control over engagement parameters. This coincided with the capabilities of the original missile. A number of subsequent improvements in the missile have, however, outstripped the ability of the HSCLCS to accommodate them.

While the 1981 modification partially corrects this situation, it does so at the expense of complicating the parameter entry process to the point where the probability of error is, again, high. The potential for future enhancements using the same basic system is small.

C. PROPOSED SOLUTION

The deficiencies outlined above have been considered by responsible DOD authorities and development of a further modification to the HSCLCS is now in process under the sponsorship of the Naval Sea Systems Command. The changes involve replacement of the existing WCIP with a new version containing a display, a discrete switch and indicator panel and a local processor. Other than software changes to the DPC to accommodate the new panel, only minor modifications to the remainder of the system are involved. Figure 5.3 is a pictorial representation of the new WCIP.

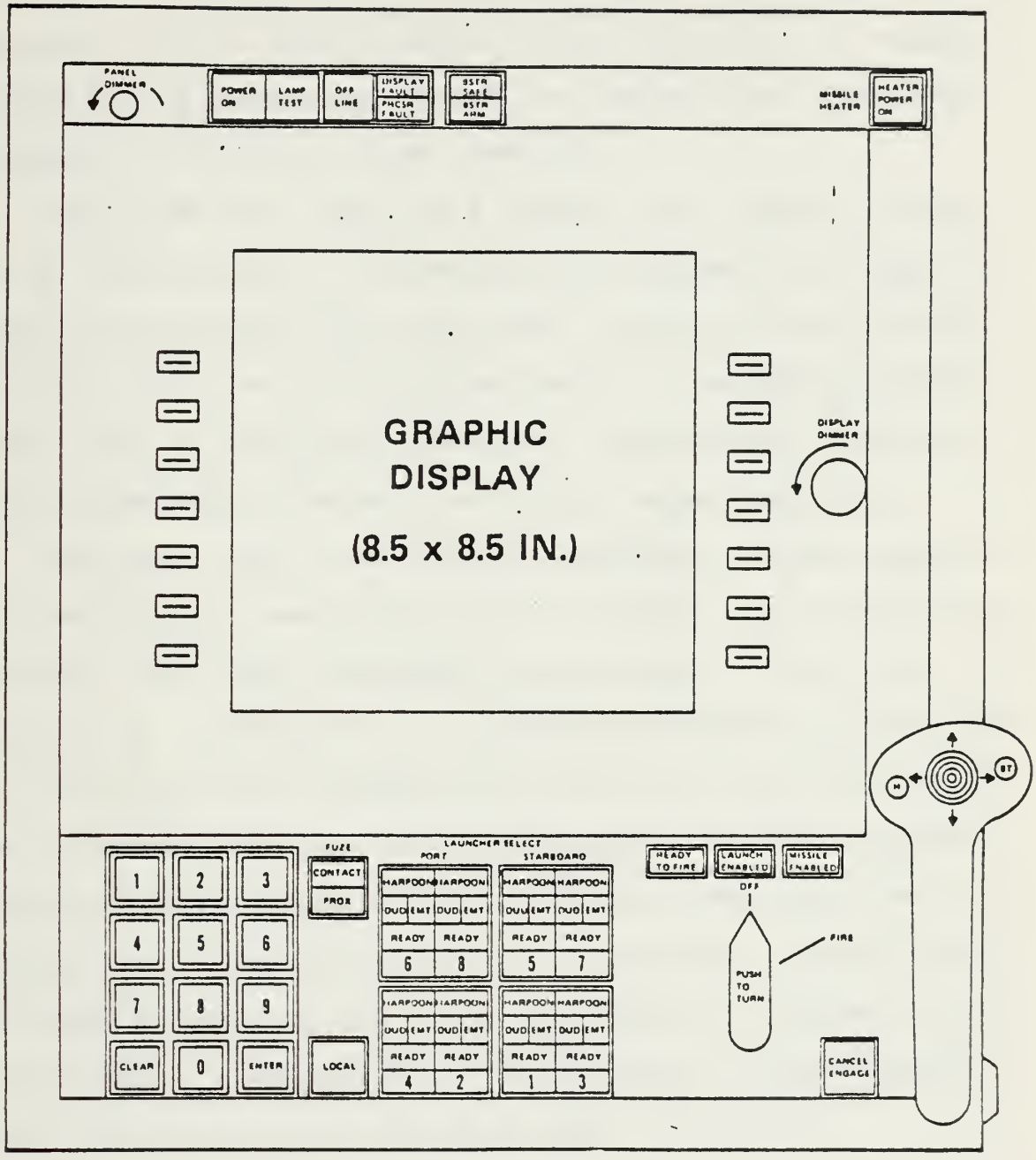

Figure 5.3
Proposed Replacement WCIP

The panel will provide a graphic display of a subset of the tactical situation to a maximum range of 128 miles. The operator will be able to enter tracks (positions of surface contacts with associated course and speed data) which will be kept in position by dead reckoning.

Selection of a track as a target will initiate provision of information on the acquisition probability (P_{acq}) of both the selected target and other tracks in the vicinity. This data will be updated dynamically as engagement parameters, such as way point positions, are altered, allowing a trained operator to develop an optimal engagement plan.

The panel will also provide assistance to the operator in planning and conducting salvo firings and coordinated firings with other Harpoon fitted ships. This will be achieved by allowing four simultaneous engagement plans and by calculation and display of time over target for each. The principal advantage of the new panel is that by improving the human interface to the system, ambiguity and the consequent probability of error is reduced. The fact that the panel's functions are to be implemented in software also allows future enhancements, to accommodate missile improvements and more closely integrate NTDS and Harpoon.

D. REQUIREMENTS DOCUMENT

A skeletal requirement document using the NRL approach was generated for the Replacement WCIP. In order to reduce the scope of this work to a manageable level, only the

external interface was specified--the second interface, to the HCC, has been studied informally and appears to be suitable for the application of the same techniques. The narrative description discussed below and a selection of the remainder of the document is reproduced at Appendix A.

Because of the embryo nature of the replacement WCIP project, it was necessary to take the conceptual design at a given point in its development and carry on independently from there. Since the object of the study was to assess the value of the NRL techniques rather than to develop a viable design for the WCIP, this is unimportant. However, it did give rise to some difficulties which are discussed later.

1. Narrative Description

It was found necessary to generate a narrative description of the interaction between the WCIP and the human operator before attempting to start work on development of the formal requirements. This was not necessary in the case of the A-7E project, due the existence of both the previous OFP and its accompanying documentation. In chapter IV, the omission of explicit provision for communicating with the prospective users of a system was discussed as a drawback of the NRL approach. The narrative description, however, fills this role satisfactorily. The usual objections to informal system descriptions--ambiguity, inconsistency and completeness--are partly nullified by the

fact that such problems rapidly come to light when the document is used to develop the formal specification.

2. Use of Automated Aids

Generation of any substantial document requires a considerable amount of drafting and re-drafting. The techniques used for the requirement document also require much cross referencing between tables. The NRL project used no specifically designed automated aids and this is one of the advantages of the system as compared to, say, SREM which requires a substantial software investment in its own right. Fortunately a number of inexpensive general purpose tools are now becoming available on mini- and microcomputer based systems which are well within the reach of any organization likely to be developing software requirements documents. A commercial word-processing program was used during the Harpoon study and it became clear that for a more complete study or a larger project, a small data-base management system would also greatly improve productivity.

E. PROBLEMS ENCOUNTERED

As expected, a number of problems were encountered during the study. Many of these were due to inexperience but two raise issues concerning the applicability of the NRL techniques to different systems. These are discussed in this section.

1. Coping with Undefined Interfaces

As discussed in Chapter II, subsystem software requirements generation must frequently proceed in parallel with overall system architectural design--the partitioning of system functions between the various hardware and software subsystems. Although this was not a problem with the A-7E project, difficulties were encountered during the Harpoon case study due to inadequately defined interfaces.

An example of the problem is the interface between the display and the WCIP software. Since the details of the display are not yet known, the interface must be specified arbitrarily. Differences between the specified interface and the real one implemented by the hardware must be catered for by interface module specifications which can be defined once the hardware details are finalized. In selecting the arbitrary interface, the aim is to ensure that any likely hardware decisions can be catered for by the interface modules without changes to the core document. This requires the interface to have two properties:

a. It must be abstract--details which may vary with the device selected must be excluded.

b. It must be 'high-level' enough to minimize the possibility that it will intersect the physical hardware interface. For example, an assumption that the WCIP system software reads the position of the balltab or joystick and uses this to con-

trol the position of the cursor on the screen will fail if cursor positioning is controlled within the display device itself.

Achieving the second property calls for fine judgement. Correction of the core document to alter a specification which intersects the hardware interface may well require major changes. On the other hand, too high a level of interface will require in the specification of internal rather than external behaviour and will result in a partial implementation definition.

Sometimes this will be unavoidable. For example a display with its own processor--by no means uncommon--may well be able to accept high level commands to display the geometrical figures required for the WCIP such as special symbols, uncertainty elipses, and engagement lines. For simpler displays, the system software will be required to break down the patterns into displayable elements such as pixels or vectors.

Such software would not meet the requirements of a single abstract interface module [26] since it would require changes both when the display changed and when the desired patterns changed. This can be avoided by two modules - one to translate the patterns into abstract elements such as lines, arcs and pixels and one to map these elements to the actual display capabilities. Changes in the patterns would

then require changes only to the first module and changes in the display hardware only to the second.

Because of the possibility that the first type of display may be chosen, the requirements document must use the first approach. The fact that this choice will constrain the implementation if the simpler display is selected must be accepted.

2. Handling General Purpose Devices

The A-7E software is required to interface with a number of hardware devices. With one exception, these are special purpose in nature and are controlled by a small number of signals. The approach of specifying input and output data items in terms of their hardware effects and independent of software functions works well in such cases.

The exception is the computer display panel, which is constructed of fourteen 7-segment numerical displays and is used for a number of functions. As the A-7 Requirements Document notes, the approach above fails in this case due to the huge number of possible output signals (2^7 for each of 14 windows) and the fact that only a small subset of these has any meaning [27:4-65]

In the case of the computer display panel, the NRL team made two changes to the approach used for other hardware interfaces. First, the panel display functions were related to 'semantic entities' such as present position or wind speed and direction rather than directly to the output

data item of a particular bit pattern controlling the segments of the display. These entities are determined by the OFP sofware rather than the hardware interface. The second change was to specify a virtual panel for each of the functions of the real panel. Each software function which produced a panel output could therefore be assumed to have its own dedicated panel. A separate specification section provided rules for determining which of the panel display functions controls the real panel at any given time.

The interfaces of the WCIP software, in contrast with those of the A-7E OFP, are almost all general purpose in nature. The only interface which can be successfully specified in terms of software functions acting on hardware data output items is that to the switch/indicator devices of the firing sequence panel.

The data entry/display panel is a considerably more flexible device than the A-7E computer display panel but can be handled in much the same way. Tying software functions to semantic entities such as a particular output message seems intuitively more sensible than cluttering the specification with the mass of detail required to convert these messages into their particular hardware representation. Similarly the use of virtual buttons and panels simplifies the document by segregating the details of mapping virtual to real devices to a single section. Although the WCIP/HCC interface was not considered in the study, the same

techniques should be suitable for this and other computer-to-computer interfaces.

More severe difficulties arise with the tactical display since this is an even more flexible device. For the WCIP, the same approach of relating functions to semantic entities was used--in this case, track symbols, flight paths, uncertainty elipses etc. This can be done since the number of these entities displayable is limited and small. In bigger systems, for example NTDS, there are a very large number of different symbols and the number which can be displayed simultaneously is not limited. It is difficult to see how the NRL approach could be used effectively in this context without introducing excessive complexity.

F. IMPRESSIONS

During the study of the NRL techniques and their application to the WCIP software requirements, a number of impressions were gained. These are offered below and represent the writer's opinions only:

1. Complexity

The criticism discussed in chapter IV--that the state-machine technique does not permit decomposition of complexity--is at least partially true. The A-7E OFP is a small program as embedded systems go. With a larger program, the monolithic nature of its view of the system described would make it difficult both to generate and to analyse.

Work along the lines of the approach used by PAISLey (chapter 3) may provide a solution to this problem. In PAISLEY, the system is modeled as a group of interacting processes, themselves described using the state machine technique. Such an organization could serve to contain the complexity of a large system at the expense of encroachment of the specification on the top level of architectural design.

2. Verifiability

Because of its emphasis on specifying external behaviour, the method lends itself readily to the generation of system testing procedures. This is because the response of the system to conditions and events is directly specified rather than having to be inferred from a model as it is in the systems discussed in Chapter 3. If the specification techniques were made more rigorous it should, in principle, be possible to generate testing procedures automatically but this may detract from the ability of the method to adapt to situations such as the specification of functions concerned with general purpose devices discussed earlier.

3. Overall System Design

Earlier, the requirement to generate a narrative system description of the human interface to the system was discussed. This is a system level design function which the requirements document must reflect as opposed to a software design function which should be left until implementation

The human interface can be looked at as comprising two parts--hardware and procedural. The hardware interface to individual devices such as the joystick, buttons and screen is usually well defined and is something to which the software must accommodate. In contrast with this, the procedural details of how the human will interact with the devices are usually controlled directly by the software, and are frequently poorly defined at the system design stage. The result, as in this case, is that their design and specification, by default, becomes the task of the software requirements writer.

An attempt was made during the Harpoon study, to use the NRL techniques for this purpose but it rapidly became clear that they are more suited to documenting external system behaviour than to developing it. The reason for this is that the tabular presentation is excellent for answering questions about what conditions are required for a particular output but less useful for providing a mental picture of system response to inputs. As a result, the narrative system description mentioned earlier was used.

Since the narrative is useful for communication with the user and forms a nucleus of the operations manual--a necessary part of the documentation of any system--this approach seems to be viable. Requirement techniques which allow simulation, however, such as SREM and PAISLey appear

to offer better promise in this regard since they would allow the human interface to be developed iteratively in conjunction with the end users.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The techniques developed by NRL for the A-7E OFP project appear to meet the goals which were set for them. Among the strengths of the techniques are the following:

a. The almost complete absence of intrusion of the resulting document on implementation detail.

b. The fact that no major hardware, software or training investment is required to use them.

c. The ease with which test specifications may be developed from the requirements document.

The method also proved relatively straightforward to apply to another embedded system which, although small, differed markedly in technology and intent from the one for which it was developed. While not all aspects of the new system could be conveniently described using the exact approach applied to the A-7E software, it was easy to adapt the appropriate techniques to the new circumstances, resulting in a document which retained the style and rigour of the original.

Among the apparent weaknesses of the method are:

a. Its inability to cater easily for different levels of abstraction or to provide other means for communicating a mental picture of system operation as

well as a rigorous specification against which such a picture can be tested.

b. The fact that handling of and interfaces not fully defined can only be done at the expense of intrusion of the specification into the architectural design stage of the implementation.

c. The artificiality necessary to handle multi-purpose output devices adequately and the lack of a technique for handling general purpose devices such as CRT displays.

These qualities suggest that the techniques are best applied initially to small and medium scale systems where investment in more hardware or software intensive systems is not justified and where the lack of abstraction is not too great a disadvantage.

Since development of the requirements does not involve preliminary architectural design of the software and the specification uses no formal programming language, expertise in computer science need not be a pre-requisite to use of the techniques. The NRL approach is thus better suited than others such as SREM for direct involvement by experts in the domain of the system under design rather than in software design.

B. RECOMMENDATIONS FOR FURTHER RESEARCH

Clearly, further experience in the use of the method is necessary to evaluate its potential more fully. The ongoing experiment by Basili and Weiss, discussed in chapter 4 should provide interesting data but will not replace the need for assessment in different applications, preferably by personnel who have first hand experience of other systems such as SREM. Among the aspects of the method which warrant further attention are those discussed as weaknesses above -- particularly the issues of handling complexity and incorporation of general purpose devices.

In the more general area of software requirements analysis, work is required to clarify further the interactions between the specification and design phases described in chapter 2. Further development of the work reported in [8] could lead to the development of guidelines covering this aspect of software acquisition.

Finally, the Harpoon system discussed in chapter 5 offers a fruitful source of research projects. The system is large enough to provide a challenge and produce worthwhile data but not so large as to be unmanageable or excessively expensive. There are a number of projects in the Software Engineering and related fields which would profit from using this system as a test bench.

APPENDIX A

This Appendix contains a narrative description of the software functions for the proposed Harpoon replacement Weapons Control Indicator Panel, together with a selection of the more important interface descriptions, mode tables and function descriptions. Most of the information is in a skeletal form since the objective of the study was to investigate the application of the NRL techniques rather than generate a working requirements document.

NARRATIVE SYSTEM DESCRIPTION

This section provides an informal (and incomplete) narrative description of the operation of the engagement control system. It is biased towards system operation as perceived by the operator, concentrating on information displayed and allowable actions which may be taken.

To simplify the description, four modes of system operation are defined: Display, Idle, Designate and Engage. This is an arbitrary division but serves to organize the document since the modes differ from one another principaly in the information displayed and the allowable actions of the operator.

For each mode, the following information is provided:

* brief description
* rules governing mode entry/exit
* rules governing actions/displays applicable in the mode information displayed
* allowable actions

DISPLAY MODE

Mode Description:

The system is always in display mode while the HWS is operational. The mode is specified to allow for future non-operational modes such as test or training. In this mode, the system maintains and updates a display of the tactical situation which consists principaly of track symbols indicating the positions of contacts within the range scale of the display.

Tracks may be entered manually by the operator or automatically via an interface to the NTDS system. A maximum of 12 tracks may be maintained by the system, including the track designating own ship and one aircraft track. Tracks are entered using own ship relative coordinates of range and bearing.

Track Selection:

The operator is provided with a thumb operated control column which can be manipulated to position a cursor on the screen. Use of this control together with adjacent 'hook' and 'break track' pushbuttons allow any track to be selected for special attention. Such a track is said to be hooked. Only one track may be hooked at any one time.

Information Display:

Tracks are displayed using standard NTDS symbology for surface tracks. Each track is displayed with a fixed length course leader.

Track positions are periodically updated for own ship motion by dead reckoning. The operator is responsible for updating the positions of tracks for their own motion.

The following HWS and status information is displayed:

* system status (on/off/fault)
* missile inventory
* time of day

The following information is displayed for a hooked track:

* range
* bearing

Allowable Actions:

The following actions may be taken at any time:

* enter manual track
* select range scale
* offset display
* set display configuration
* enter environmental data

The following actions may be taken when a track is hooked:

* drop (delete from system)
* reposition
* enter course

IDLE MODE

Mode Description:

The system is in Idle mode at any time when it is in Display mode and in neither Designate nor Engage mode.

Allowable Actions:

In Idle mode, a hooked track may be designated to the HWS for attack.

DESIGNATE MODE

Mode Description:

The system is in designate mode at any time when a track has been designated to the HWS for possible attack by one or more missiles. It ceases to be in designate mode on receipt of ITL for the last missile or on release of the designated track. In this mode, the system accepts and displays engagement data and presents this together with acquisition probabilities and other assessment information.

Only one target may be designated at any given time but one to four missiles may be selected to engage that target simultaneously, using independent engagement plans.

An attempt to designate a track which is unengageable as a result of over or under range results in an operator alert. The system remains in its preceding mode(s).

Information Display:

The following information is displayed for a hooked track in designate mode

* Pacq

The following information is displayed for the designated track

* Selected Time on Target
* Time to Launch first missile

The following status information is displayed in designate mode for each missile in the planned engagement

* Pacq of designated track
* missile search mode
* search pattern size (RBL)
* min/destruct range (BOL)
* search priority
* attack mode
* time to launch

The following information is displayed graphically in designate mode. This information refers to the most recently selected missile:

* booster drop zones
* missile flight path
* search pattern
* search priority
* uncertainty elipse (Pacq = ??)

The following HWS information is displayed

* HFCS ready
* booster safe/armed
* missile(s) enabled
* heater power on
* ready to fire

Allowable Actions:

The following engagement plan actions may be taken in designate mode:

* select launcher
* select missile
* select time on target
* enter/delete/reposition way point
* change missile mode
* change search pattern size (RBL)
* change min/destruct range (BOL)
* change search priority
* change attack mode
* set flyout range

The following engagement control actions may be taken:

* cancel engagement
* apply/remove heater power
* arm/disarm boosters
* fire

ENGAGE MODE

Mode Description:

The system is in engage mode whenever any missile is in flight. This is defined as the period from the receipt of ITL to (TBC) seconds after the projected time to impact reaches zero. The system may be in engage and designate mode simultaneously. On completion of the last in flight period the system reverts to idle mode. The system may also be forced to idle mode at any time by the operator. In engage mode, additional information on the engagement is displayed.

Information Display:

The following status information is displayed in engage mode:

* information displayed in designate mode
* time to go to impact

The following information is displayed graphically for each engaged track.

* information displayed in designate mode
* missile position (by dead reckoning)

Allowable Actions:

* cancel engagement (note that this does not affect a missile once it is launched - all it does is revert the system to idle mode)

SOFTWARE INTERFACES

This section defines the physical interfaces of the system software. These interfaces are described with reference to the hardware device interfaces:

a. The HCC interface.
b. The display console.

The interface definitions comprise a brief description of the external device capabilities and a list of all applicable input and output data items. Input items are those originating at the external device, output are those routed to the external device.

Initially, only a listing of the mnemonic names of the data items is given. Later expansion will be required to cover applicable ranges of values, resolution and data representation.

DISPLAY DEVICE

The display device is a general purpose CRT or plasma display console with the capability of displaying both graphics and text. The device is not yet fully specified but incorporates the following input and output capabilities:

a. A screen area for displaying graphics.
b. A screen area for displaying text.
c. A joystick capable of controlling the position of a cursor on the screen.
d. A number of general purpose buttons with legends alterable under software control.
e. A keypad for numeric entry.
f. Special purpose switches and indicators for launcher selection/status, firing sequence, etc.

For the purposes of this document, the device is considered as a number of virtual devices, a tactical display, comprising the majority of the display screen and the joystick, a data entry/display device comprising the peripheriphery of the screen together with the general purpose input buttons, and a firing panel, comprising the dedicated switch/indicator devices.

Tactical Display Device

The tactical display device uses a general purpose CRT or plasma panel to provide a schematic representation of the tactical situation. Its single input is the joystick which provides positioning information to allow the operator to 'select' a point on the display.

Input Data Items

/CURSOR/
A signal which indicates the position of a point on the screen selected by the operator.

/HOOK/
Signal indicating that the 'hook' button on the joystick has been pressed.

/BREAK/
Signal indicating that the 'break track' button on the joystic has been pressed.

Output Data Items

//TRACK0//..//TRACK10//
Order to display a single track symbol on the display.
Data:
- Position
- Symbol type
- Course
- Display parameters (eg. flashing)

//HOOK//
Order to display a hook circle about any track.
Data:
- Position

//TRAJECTORY//
Order to display a trajectory path between the own ship track symbol and any other track symbol. A path consists of one or more straight line segments.
Data:
- Start position
- End position
- Intermediate position(s)

//SEARCH//
Order to display a search pattern. This consists of a closed area bounded by two radial lines and two arcs.
Data:
- Origin
- Axis direction
- Min radius
- Max Radius

//ELIPSE//
Order to display an uncertainty elipse. This is a locus of constant kill probability about a target position.
Data:
- Origin
- Axis direction
- Major axis
- Minor axis

//BSTDROP//
Order to display booster drop zone. This is an area bounded by two radii and two arcs.
Data:
- Origin
- Axis direction
- Min radius
- Max Radius

Data Display

This virtual device consists of a display panel (part of the screen of the Tactical Display), a numeric entry panel and a number of general purpose input buttons.

For the remainder of this document, the display panel is considered to be broken up into sections, each one dedicated to a particular display item and the buttons are each considered to be dedicated to a single input quantity. In fact, this is not the case and information regarding the ovelaying of these virtual functions onto the limited number of physical display sections and buttons is given below.

Logical/Physical Mapping:

The data display/entry device contains 14 Variable Action Buttons (VAB), the legend which describes the function of each button is written on the edge of the screen adjacent to the button. The area used for the legend may be highlighted by the use of reverse video to provide a 'cue light' for feedback to the operator or for other purposes.

This arrangement can be used to provide a multi-level menu selection system. A selection of tables describing the individual menus follows.

MENU #: 0 (main menu)
MENU LEVEL: 0
REACHED FROM:
VALID IN MODE: All

VAB	LEGEND	FEEDBACK	ACTION	NEXT MENU
1	16 mile	!select1!	/R16/	
2	32 mile	!select1!	/R32/	
3	64 mile	!select1!	/R64/	
4	128 mile	!select1!	/R128/	
5				
6				
7				
8	set time	!cue!	/TIME/	
9	set env	!flash!	/ENVDATA/	2
10	new trk	!flash!	/NEWTRK/	3
11	repos	!flash!	/REPOS/	
12	drop trk	!flash!	/DRPTRK/	
13	classify	!flash!	/CLASS/	3
14	desig	!flash!	/DESIG/	4

MENU #: 1 (environmental data)
MENU LEVEL: 1
REACHED FROM: menu 0
VALID IN MODE: All

VAB	LEGEND	FEEDBACK	ACTION	NEXT MENU
1				
2				
3				
4				
5				
6				
7	tac disply	!flash!	/MAIN/	0
8	wind speed	!cue!	/WIND/	
9	wind dirn	!cue!	/WDIR/	
10	sea state	!cue!	/SEA/	
11	rain	!select1!	/RAIN/	
12	dry	!select1!	/DRY/	
13				
14				

MENU #: 4 (attack menu)
MENU LEVEL: 1
REACHED FROM: main menu
VALID IN MODE: *desig*, *engage*

VAB	LEGEND	FEEDBACK	ACTION	SUB-MENU
1	stbd lchr	!select1!	/SLCHR0/	
2	port lchr	!select1!	/SLCHR1/	
3	cell 1	!selectn!	/CELL1/	
4	cell 2	!selectn!	/CELL2/	
5	cell 3	!selectn!	/CELL3/	
6	cell 4	!selectn!	/CELL4/	
7	tac disply	!flash!	/MAIN/	0
8	srch mode	!flash!	/SEARCH/	5
9	srch pri	!flash!	/SPRI/	6
10	attack md	!flash!	/ATTMD/	7
11	waypoint	!cue!	/WAYPT/	
12	time at tgt	!cue!	/TOT/	
13	fly out r	!cue!	/FOR/	
14	pacq	!flash!	/PACQ/	

Other Input Items:

/VALUE/
Numeric value entered by the operator using the keypad

Output Data Items:

//VALUE//
General purpose numerical output

//TOD//
Time of Day
hhmmss zulu time

//TOT//
Time on Target
hhmmss zulu time

//HRANGE//
Range of hooked track

//HBRG//
Bearing of hooked track

//HPACQ//
Probability of acquisition of hooked track

//ENGMSG1// . . //ENGMSG4//
Engagement message for missile 1..4
data: cell
fcs status
acquisition probability
search mode
search priority
attack mode
launch time
time to impact

//DMINRG//
Minimum BOL range for designated track

//DDRG//
Destruct range for designated track

//DFOR//
Fly out range for designated track

//DTRAJ//
Terminal Trajectory for designated track

Firing Panel

The firing panel consists of the dedicated switch/indicator devices on the WCIP other than the joystic assembly and the numeric keypad which are included in the tactical and data displays respectively.

Input Data Items:

/LCHRP/ . . /LCHRS/
Select port/starboard launcher

/CELL1/ . . /CELL8/
Select cell 1..8

/HEATER/
Select missile heaters on/off

/CANCEL/
Cancel engagement

/FIRE/
Initiate firing

/FUZE/
Select fuze mode (contact/proximity)

/ARM/
Arm/disarm booster

/LOCAL/
Select local/normal mode

Output Data Items:

//CSTAT1// . . //CSTAT8//
Missile Cell Status 1..8
Ready/Dud/Empty

//HTRPWR//
Heater power on

//RTF//
Ready to fire

//LCHEN//
Launcher enabled

//MSLEN//
Missile enabled

//ARMED//
Booster safe/armed

//FUZE//
Contact/proximity fuze

//LOCAL//
System Status (local/normal)

MODE TRANSITION TABLE

The table below shows the events which cause transitions between modes. Entries in the table refer to the list of events below. Blank entries indicate that the corresponding transition is not allowed. In addition to the modes listed, the system is always in *display* mode.

Exited Mode:				
idle		1		
designate	2		3	4
designate AND *engage*	5	6		7
engage	8			
Entered Mode:	***idle***	***designate***	***designate* AND *engage***	***engage***

Events causing transitions:

1. @T(/DESIG/) AND !desig valid!

2. @T(/CANCEL/) OR @T(NOT !desig valid!)

3. @T(/ITL/) AND (!msls designated! > 1)

4. @T(/ITL/) AND (!msls designated! = 1)

5. @T(/CANCEL/)

6. @T(!msls inflight! = 0)

7. @T(/ITL/) AND (!msls designated! = 1) OR
   @T(NOT !desig valid!)

8. @T(/CANCEL/) OR @T(!msls inflight! = 0)

FUNCTION DESCRIPTIONS

Periodic Function Name: update tracks for ownship movement

Modes in which function required: *display*

Initiation/Termination Events: @T(in mode)/@T(NOT(in mode))

Output Data Item: //TRACK1// . . //TRACK10//

Description: The position of each allocated track symbol is periodically altered to compensate for ownship movement since the previous correction. Because the incremental movement is small, no adjustment of the correction for newly entered tracks is applied.

Condition Table

MODES	CONDITIONS	
display	!track allocated!	X
//TRACKn//	position = position - !ownship movement!	position unchanged

Demand Function Name: hook track

Modes in which function required: *display*

Output Data Item: //HOOK//

Description: The operator may hook a track by positioning the cursor 'close' to the track and depressing the 'hook' button. This causes the hook symbol to be displayed coincident with the track. The hook symbol is always displayed and defaults to own ship position

Condition Table

MODES	EVENTS	
display	@T(/HOOK/ = on) AND !correlated!	@T(in mode) OR @T(/DPTRK/ = on)
//HOOK//	position = position of !correlated track!	position = position of //TRACK0//

Demand Function Name: display hooked track info

Modes in which function required: all

Output Data Item: //HRANGE//,//HBRG//,//HPACQ//

Description: bearing and range are displayed for the hooked track in all modes, Pacq is displayed in *designate* mode

Event Table

MODES	CONDITIONS	
display	@T(/HOOK/ = on) AND !correlated!	X
desig	X	@T(/HOOK/ = on) AND !correlated!
//HRANGE//	!range of hooked track!	!range of hooked track!
//HBRG//	!brg of hooked track!	!brg of hooked track!
//HPACQ//	not displayed	!pacq of hooked track!

DICTIONARY

!cue!	response to menu selection - cue light remains on until associated entry action complete
!correlated!	distance from cursor position to closest track less than (TBD)/range scale
!correlated track!	track closest to cursor position when !correlated! true.
!desig valid!	indicates that a designation is valid, ie: (//HRANGE// >= !harpoon minimum range!) AND (//HRANGE// >= !harpoon maximum range!)
!flash!	response to menu selection - cue light flashes for ?? ms. NB: not required if VAB buttons provide tactile feedback
!msls designated!	number of missiles selected - number of missiles fired since entering *designate* mode
!msls inflight!	number of missiles fired - number of missiles attacked since entering *engage* mode
!select1!	response to menu selection - cue light of most recently selected menu item remains on
!selectn!	response to menu selection - cue light of each selected menu item remains on

LIST OF REFERENCES

1. Patterson, J.C., "Real Time Multiprocessing may depend on Ada Software", Defense Electronics, Vol 13 No 8, August 1981, pp 82-85.

2. De Roze, B.C. and Nyman, T.H., "The Software Life Cycle - A Management and Technologiucal Challenge in the Department of Defense", IEEE Transactions on Software Engineering, Vol SE-4 No 4, July 1978, pp 309-318.

3. Hoare, C.A.R., "Software Engineering: a Keynote Address". Proceedings of the 3rd International Conference on Software Engineering, IEEE, 10-12 May 1978.

4. Thayer, R.H., Pystler, A.B. and Wood, R.C., "Major Issues in Software Engineering Project Management". IEEE Transactions on Software Engineering, Vol SE-7 No 4, July 1981, pp 333-342.

5. Lehman, J.H., "How Software Projects are Really Managed". Datamation January 1979, pp 119-129.

6. Boehm, B.W., McClean, R.K. and Urfrig, D.B., Some Experience with Automated Aids to the Design of Large-Scale Reliable Software. TRW Report TRW-SS-75-01, TRW Inc, Redondo Beach California. January 1975.

7. Freeman, P., "The Context of Design", in Freeman P and Wasserman A I. Tutorial on Software Design Techniques. IEEE Computer Society, Long Beach, California, 1980.

8. Mc Henry, R.C. and Walston, C.E., "Software Life Cycle Management: Weapons Process Developer", IEEE Transactions on Software Engineering, Vol SE-4 No 4, July 1978, pp 309-318.

9. Disatso, J.R., "Software Management - A survey of the Practice in 1980", Proceedings of the IEEE, Vol 68, No 9, September 1980, pp 1103-1119.

10. Yeh, R.T. and Zave, P., "Specifying Software Requirements". Proceedings of the IEEE, Vol 68, No 9, September 1980, pp 1077-1085.

11. Mullery, G.P., "CORE - A Method for Controlled Requirement Specification". Proceedings 4th Intl Conference on Software Engineering pp 126-135, IEEE 1979.

12. Heninger, K.L., "Specifying Software Requirements for Complex Systems: New Techniques and their Application", IEEE Transactions on Software Engineering, Vol SE-6, No 1, January 1980, pp 2-13.

13. Balzer, R. and Goldman, N., Principles of Good Software Specification and their Implications for Specification Language, Proceedings Conference on Specifications of Reliable Software, pp 58-67, IEEE 1979.

14. Davis, C.G. and Vick, C.R., "The Software Development System". IEEE Transactions on Software Engineering, Vol SE-3 No 1, January 1977, pp 69-84.

15. Department of Defense Military Standard 1679, Weapons System Software Development, 1 December 1978.

16. Alford, M.W., "A requirements Engineering Methodology for Real Time Processing Requirements". IEEE Transactions on Software Engineering, Vol SE-1 No 1, January 1977, pp 60-69.

17. Bohm, C. and Jacopini, G., "Flow Diagrams, Turing Machines and Languages with only two Formulation Rules". Communications of the ACM, Vol 9, No 5, May 1966, pp 366-371.

18. Bell, T.E., Bixler, D.C. and Dyer, M.E., "An Extendable Approach to Computer-aided Software Requirements Engineering". IEEE Transactions of Software Engineering, Vol SE-3 No 1, January 1977, pp 49-69.

19. Nam, C.W., Software Requirements Engineering: Experience and New Techniques. Ph.D. Thesis, University of California, Berkeley, 1981.

20. Teichroew, D. and Hershey, E.A., "PSL/PSA: a Computer-aided Technique for Structured Documentation and Analysis of Information Systems". IEEE Transactions on Software Engineering, Vol SE-3 No 1, January 1977, pp 41-49.

21. Yamamoto, Y., An Approach to the Generation of Software Life Cycle Support Systems, Ph.D Thesis, University of Michigan, 1981.

22. Zave, P., "An Operational Approach to Requirements Specification for Embedded Systems". IEEE Transactions on Software Engineering, Vol SE-8 No 3, May 1982, pp 250-269.

23. Tse, T.H. and Pong, L., "A Review of System Development Systems", The Australian Computer Journal, Volume 14 No 3, August 1982, pp 99-109.

24. Ross, D.T., "Structured Analysis (SA): A Language for Communicating Ideas". IEEE Transactions on Software Engineering, Vol SE-3 No 1, January 1977, pp 16-34.

25. Ross, D.T. and Schoman, K.E., "Structured Analysis for Requirements Definition". IEEE Transactions on Software Engineering, Vol SE-3 No 1, January 1977, pp 6-15.

26. Parker, R.A., Heninger, K.L., Parnas, D.L. and Shore, J.E., Abstract Interface Specifications for the A-7E Device Interface Module. Naval Research Laboratory memorandum Report 4385, November 20 1980.

27. Henninger, K.L., Kallander, J.W., Shore, J.E. and Parnas, D.L., Software Requirements for the A-7E Aircraft. Naval Research Laboratory Memorandum Report 3876, November 27 1978.

28. Boehm, B.W., et al, Characteristics of Software Quality North-Holland, 1978.

29. Parnas, D.L., "A Technique for Software Module Specification with Examples". Communications of the ACM, Volume 15 No 5, May 1972, pp 330-336.

30. Liskov, B. and Zilles, S., "An Introduction to Formal Specifications of Data Abstractions". In Yeh, R.T. (Ed) Current Trends in Programming Methodology Prentice Hall, 1977.

31. Basili, V.R. and Weiss, D.M., "Evaluation of a Software Requirements Document by Analysis of Change Data". 5th International Conference on Software Engineering, IEEE, March 9-12 1981.

INITIAL DISTRIBUTION LIST

		No of Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 54 Department of Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
4.	Acquisition Library, Code 54Al Naval Postgraduate School Monterey, California 93940	1
5.	Professor Norman R. Lyons, Code 54Lb Department of Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
6.	LCDR Ronald M. Modes, USN, Code 52Mf Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
7.	Commander N. D. H. Hammond, RAN Director Naval Project Coordination Navy Office, Canberra 2600 AUSTRALIA	2

200046

Thesis
H17535 Hammond
c.1 Requirements analysis and specification methodologies for embedded computer systems: survey and case study.